

# P1-63,64,65: Schaltlogik Vorbereitungshilfe

Christian Barth (1196701), Christian Benz (1204880)

24.01.2008

## Inhaltsverzeichnis

<b>0</b>	<b>Einleitung</b>	<b>2</b>
0.1	Das Experimentierboard . . . . .	2
<b>1</b>	<b>Gatter aus diskreten Bauelementen</b>	<b>3</b>
1.1	AND-Gatter . . . . .	3
1.2	NOT-Gatter . . . . .	5
1.3	OR-Gatter . . . . .	6
<b>2</b>	<b>Weitere einfache logische Funktionen (Gatter), realisiert mit ICs</b>	<b>8</b>
2.1	Inverter (NOT-Gatter) aus NAND- oder NOR-Gatter . . . . .	10
2.2	XOR-Gatter . . . . .	10
2.3	XOR-Gatter aus NAND-Gatter . . . . .	11
<b>3</b>	<b>Addierer</b>	<b>12</b>
3.1	Halbaddierer . . . . .	12
3.2	Volladdierer . . . . .	13
3.3	Subtrahierer . . . . .	15
<b>4</b>	<b>Speicherelemente</b>	<b>18</b>
4.1	RS-Flip-Flop (RS-FF) . . . . .	19
4.2	Getaktetes RS-Flip-Flop (RST-FF) . . . . .	20
4.3	Jump-Kill-Master-Slave-Flip-Flop (JK-MS-FF) . . . . .	23
<b>5</b>	<b>Schieben, Multiplizieren und Rotieren</b>	<b>25</b>
5.1	4-Bit-Schieberegister . . . . .	26
5.2	4-Bit-Rotationsregister . . . . .	26
<b>6</b>	<b>Zähler</b>	<b>28</b>
6.1	4-Bit-Asynchronzähler . . . . .	28
6.2	Asynchroner Dezimalzähler . . . . .	29
6.3	4-Bit-Synchronzähler . . . . .	30
6.4	Synchroner Dezimalzähler . . . . .	31
<b>7</b>	<b>Digital-Analog-Wandlung</b>	<b>32</b>

## 0 Einleitung

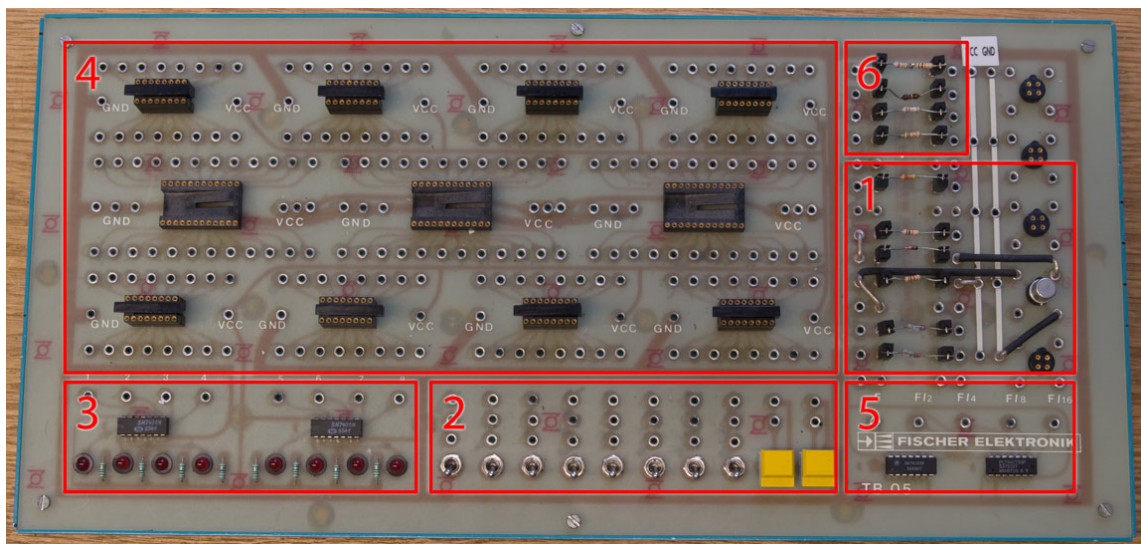
Diese Vorbereitungshilfe soll einen Überblick über den Versuch „Schaltlogik“ und deren Hintergründe geben. Sie ist nicht als Vorlage für eine Vorbereitung gedacht und dazu auch nicht geeignet, da Sie stellenweise viel zu tief ins Detail geht und andernorts auf gewisse Feinheiten gezielter Fragestellungen des Versuchs nicht näher eingeht.

Die Nummerierung der Kapitel orientiert sich an der Aufgabenstellung des Versuchs. Am Ende jedes Kapitels sind zur Überprüfung des Erlernten ein paar Fragen zusammengestellt, die man danach beantworten können sollte. Die Fragen dienen nur zur Selbstkontrolle und müssen nicht schriftlich in der Vorbereitung beantwortet werden!

Erfahrungsgemäß werden einige (teilweise offensichtliche) Fehler aus fremden Vorbereitungen von vielen Studenten besonders gerne in deren eigene Vorbereitung übernommen. Aus diesem Grund sind an manchen Stellen **Hinweise** auf diese Fehler gegeben.

### 0.1 Das Experimentierboard

In allen Versuchsteilen werden Schaltungen auf diesem Experimentierboard aufgebaut:

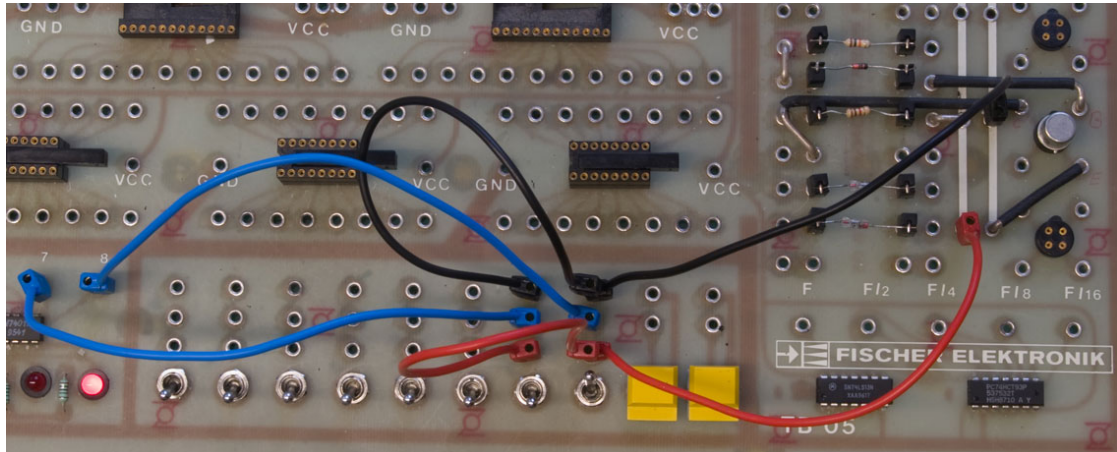


Die einzelnen Bereiche sind:

1. Diodengatter: Hier werden die AND-/NOT-/OR-Gatter aus Aufgabe 1 aufgebaut.
2. Schalter, Taster: Diese werden zum Eingeben der Zustände **1** oder **0** benötigt.
3. Leuchtdioden: Zur Anzeige der Zustände der aufgebauten Gatter/Schaltungen.
4. IC-Sockel: Hier werden die IC-Bausteine ab Aufgabe 2 eingesteckt und verschaltet.
5. Taktgeber: Über diese Ausgänge kommen Taktsignale unterschiedlicher Frequenz (abwechselnd **10101...**).

6. Widerstände: Diese werden im letzten Aufgabenteil (Analog-Digital-Wandlung) zum Umwandeln von Spannungen in Ströme benötigt.

Zum Aufbau der Diodengatter befindet sich am Ende des nächsten Kapitels eine kurze Anleitung. Zur Verwendung der Schalter: Die Signale **1** (bzw. 5 V) oder **0** (bzw. 0 V) werden über Schalter in die Eingänge der Gatter bzw. ICs eingegeben, dazu schließt man diese folgendermaßen an:



Über jedem Schalter befinden sich drei Anschlüsse. Der oberste und unterste sind die Eingänge, hier wird die Versorgungsspannung VCC/5 V (rotes Kabel) und Masse GND/0 V (schwarzes Kabel) eingesteckt. Der mittlere Anschluss (blaues Kabel) ist der Ausgang, er führt je nach Schalterstellung **0** oder **1**, wobei die Schalter antiintuitiv eingebaut wurden, d.h. steht der Schalter oben, wird das Signal am unteren Anschluss ausgegeben. (Vgl. obige Abbildung: Der rechte Schalter gibt **1** aus - die rechte Diode leuchtet.)

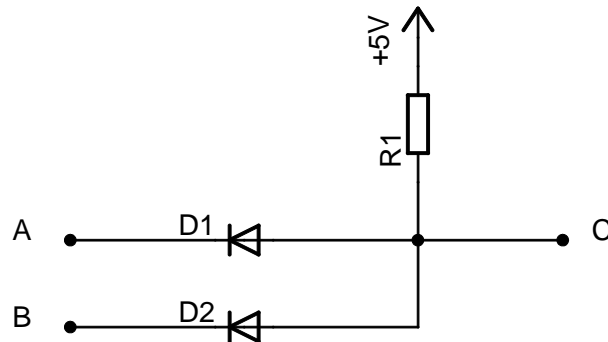
## 1 Gatter aus diskreten Bauelementen

Zum Einstieg werden logische Gatter in „Dioden-Logik“ aufgebaut. Solche Gatter haben den Vorteil, dass sie sehr wenige Bauteile benötigen, jedoch wird nach jedem Gatter die Differenz zwischen der *LOW*- und *HIGH*-Schwelle geringer, da an den Dioden jeweils ein Teil der Spannung abfällt (siehe AND- und OR-Gatter). Daher können nur sehr wenige Dioden-Gatter hintereinander geschaltet werden, bis keine Unterscheidung zwischen *LOW*- und *HIGH*-Schwelle mehr möglich ist. Aus diesem Grund wird normalerweise *TTL* („Transistor-Transistor-“) oder *CMOS* („Complementary-Metal-Oxide-Semiconductor-“) Logik verwendet. Diese beiden Logikfamilien benötigen zwar signifikant mehr Bauteile, jedoch werden hier die Schaltpotentiale durch separate Ausgangsstufen konstant gehalten.

### 1.1 AND-Gatter

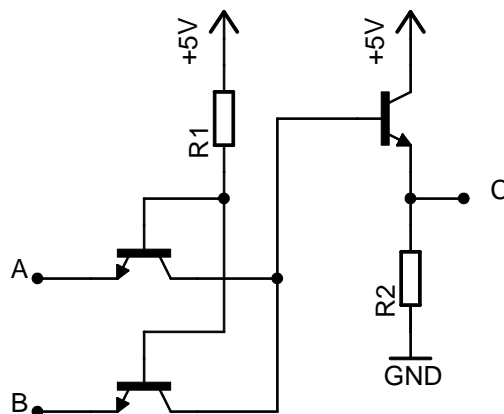
$A$	$B$	$C = A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Die logische *UND*-Verknüpfung ergibt nur dann *HIGH*, wenn alle Eingänge auf *HIGH* (**1**) liegen. In allen anderen Fällen ist das Ergebnis *LOW* (**0**). Diese Verknüpfung lässt sich als folgendes Dioden-Gatter verwirklichen:



Liegt an einem der Eingänge (A, B) **0** an, so fließt Strom durch den Widerstand und die Diode(n). In Folge beobachtet man Spannungsabfälle entsprechend deren Widerstand (Spannungsteiler). Ist der Widerstand  $R_1$  so gewählt, dass er deutlich größer als der Durchlasswiderstand der Dioden ist, so fällt fast die gesamte Spannung an ihm ab. Der Ausgang (C) ist somit auf niedrigem Potential **0**. Liegt jedoch an beiden Eingängen **1** an, so fließt kein Strom (alle Eingangspunkte der Schaltung auf gleichem Potential), es fällt daher auch keine Spannung ab und der Ausgang liegt auf **1**. Dadurch dass an den Eingängen zwei Dioden und nicht nur kleine Widerstände verwendet wurde, können Beeinflussungen zwischen den Eingängen verhindert werden: Falls ein Eingang auf **1** und der andere auf **0** liegt, ist immer eine der beiden Dioden in Sperrichtung angeschlossen und unterbindet somit den Stromfluss zwischen den beiden Eingängen.

Da der Durchlasswiderstand der Dioden größer Null ist, fällt an den Dioden auch in Durchlassrichtung ein kleiner Teil der Spannung ab, daher ist am Ausgang die *LOW*-Schwelle etwas höher als am Eingang. Schließt man mehrere dieser Gatter hintereinander so steigt die *LOW*- Schwelle immer weiter an. Folgende Schaltung wäre eine mögliche Realisierung des ANDs als TTL-Gatter, welche diese Probleme durch eine eigene Ausgangsstufe verhindert:



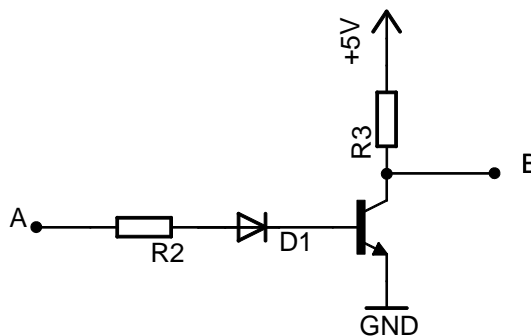
**Hinweis:** Wenn an beiden Eingängen (A,B) **1** anliegt so fließt durch die Dioden kein Strom, es fällt an ihnen daher auch keine Spannung ab. Sie werden nicht in Sperrrichtung betrieben und ihr Sperrwiderstand hat hiermit auch nichts zu tun.

An Bauteilen fallen Spannungen ab, an Ein- und Ausgängen hingegen liegen Potentiale an.

## 1.2 NOT-Gatter

A	B = $\overline{A}$
0	1
1	0

Die logische *NICHT*-Verknüpfung invertiert das Eingangssignal, eine **1** ergibt **0** und umgekehrt. Diese Verknüpfung lässt sich leicht mit einem Transistor aufbauen:



Liegt am Eingang (A) **1** an, so fließt ein Basis-Emitter-Strom. Der Transistor schaltet durch, sein Widerstand wird sehr gering. Am Spannungsteiler ( $R3$ , Transistor) überwiegt  $R3$  stark und fast die gesamte Spannung fällt an ihm ab. Der Ausgang (B) ist damit auf **0**. Liegt jedoch am Eingang **0** an, so gibt es keine Basis-Emitter-Spannung und damit fließt auch kein Basis-Emitter-Strom, der Transistor sperrt. Da der Widerstand des gesperrten Transistors sehr groß ist überwiegt am Spannungsteiler ( $R3$ , Transistor) der Transistor bei weitem und fast die gesamte Spannung fällt an diesem ab. Der Ausgang ist somit auf **1**.

Der Widerstand  $R2$  ist zur Begrenzung und Stabilisierung des Transistors: Da die Basis-Emitter-Impedanz eines Transistors stark temperaturabhängig ist, würde ohne den Widerstand  $R2$  eine kleine Temperaturerhöhung bereits einen starken Einfluss auf den Kollektorstrom und damit auf die Ausgangspotentiale haben. Wählt man  $R2$  deutlich größer als diese Basis-Emitter-Impedanz, so kann der Effekt merklich reduziert werden. In dieser Schaltung wird der Transistor als Schalter, also in Sättigung verwendet. Hierdurch ist die Leistungsaufnahme der Schaltung reduziert, gleichzeitig aber auch die Schaltzeit limitiert, da die mit Ladungsträgern geflutete Basis eine gewisse Zeit benötigt bis sie nach Abschalten des Basisstroms ladungsträgerfrei ist und der Transistor damit gesperrt wird. Die Diode  $D1$  ist eigentlich unnötig.

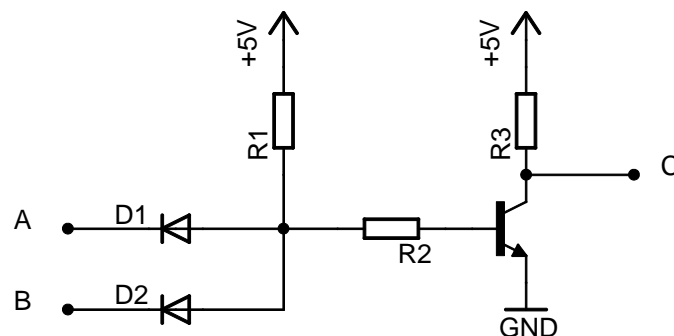
Obige Schaltung ist auf dem Experimentierboard schon komplett aufgebaut. Es müssen

nur noch die Ein- und Ausgangsleitung eingesteckt werden. In der Abbildung bei „Anschluss der Diodengatter“ ist dies durch die beiden blauen Stecker bereits geschehen.

### NAND-Gatter

A	B	$C = \overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0

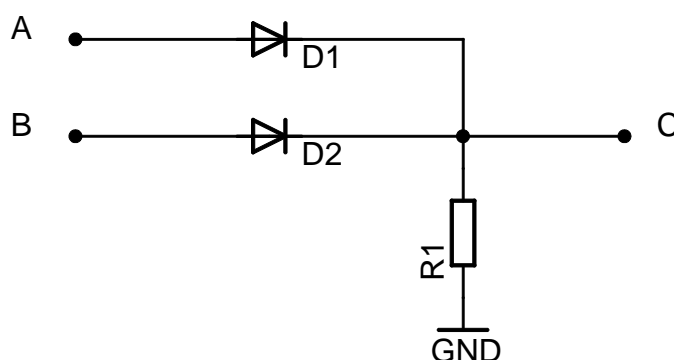
Schließt man an den Ausgang eines *AND*-Gatters ein *NOT*-Gatter an, so erhält man die *NAND*-Verknüpfung.



### 1.3 OR-Gatter

A	B	$C = A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Die logische *ODER*-Verknüpfung ergibt **1**, sobald mindestens ein Eingang auf **1** liegt.



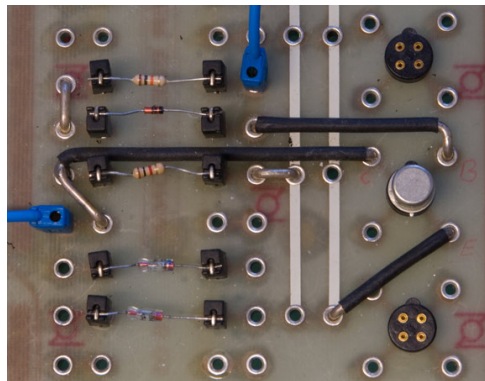
Sobald an mindestens einem der Eingänge **1** angelegt wird, fließt ein Strom durch den Spannungsteiler (Dioden,  $R1$ ). Ist der Widerstand  $R1$  sehr viel größer als der Durchlasswiderstand der Dioden gewählt, so fällt fast die gesamte Spannung an  $R1$  ab. Der


Ausgang liegt damit auf **1**. Liegt jedoch an beiden Eingängen **0** an, so sind alle Zugangsleitungen der Schaltung auf **0**. Es fließt kein Strom und der Ausgang liegt ebenfalls auf **0**. Analog zum *AND*-Gatter wird durch die Dioden an den Eingängen die Beeinflussung der Eingänge untereinander verhindert: Wenn an einem Eingang **1** und am anderen **0** anliegt, ist immer eine Diode in Sperrrichtung gepolt und unterbindet somit einen Stromfluss von einem Eingang zum anderen.

Da der Durchlasswiderstand der Dioden größer Null ist, fällt an den Dioden auch in Durchlassrichtung ein kleiner Teil der Spannung ab, daher ist am Ausgang die *HIGH*-Schwelle etwas niedriger als am Eingang. Schließt man mehrere dieser Gatter hintereinander so sinkt die *HIGH*-Schwelle immer weiter ab. **Hinweis:** Wenn an beiden Eingängen (A,B) **0** anliegt, so fließt durch die Dioden kein Strom, es fällt an ihnen daher auch keine Spannung ab. Sie werden nicht in Sperrrichtung betrieben und ihr Sperrwiderstand hat hiermit auch nichts zu tun.

### Anschluss der Diodengatter

Die gesamte Aufgabe 1 spielt sich nur in diesem Bereich des Boards ab (Schalter und LEDs einmal ausgenommen), hier sind alle erforderlichen Dioden/Transistoren vorhanden:



Die beiden Bauteile links unten sind die Dioden für das *AND*-/*OR*-Gatter. Bei genauem Hinsehen erkennt man bei der oberen Diode einen schwarzen Ring auf der rechten Seite ihres Glaskolbens. Dieser entspricht dem durchgezogenen vertikalen Balken ihres Schaltbildes  und gibt damit deren Durchlassrichtung vor (in diesem Fall nach rechts).

Der runde silberne Zylinder rechts in der Mitte ist der Transistor für das *NOT*-Gatter. Er ist wie schon erwähnt bereits komplett über die schwarz isolierten Drähte samt Schutzdiode und -Widerstand angeschlossen. Der Aus- und Eingang des *NOT*-Gatters ist durch die beiden blauen Kabel markiert.

### Fragen

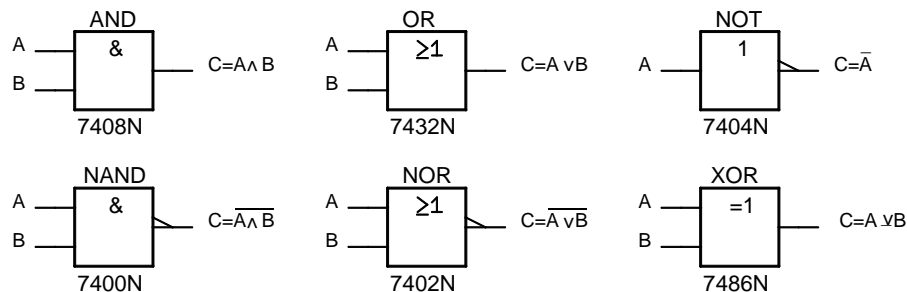
- Welche Probleme entstehen beim Hintereinanderschalten vieler solcher Diodengatter?
- Wozu werden die Dioden im *AND*-/*OR*-Gatter benötigt?

- Wie ließe sich ein AND-/OR-Gatter mit mehr als zwei (beliebig vielen) Eingängen realisieren?

## 2 Weitere einfache logische Funktionen (Gatter), realisiert mit ICs

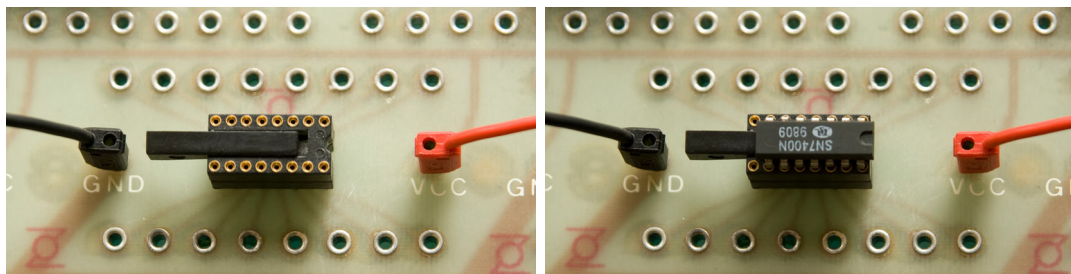
Für die nachfolgenden Schaltungen verwenden wir ICs, in welche die in Aufgabe 1 kennengelernten Logikgatter bereits integriert sind. So besteht der IC 7400 beispielsweise aus vier NAND-Gattern. An zwei seiner vierzehn Beinchen werden die Versorgungsspannung (+5V) und Masse (0V) angelegt, den restlichen zwölf Beinchen entsprechen dann direkt die acht Eingänge  $E_{11}, E_{12}..E_{41}, E_{42}$  sowie die vier Ausgänge  $A_1..A_4$ . Zu beachten ist, dass sich offen gelassene (also nicht verschaltete) Eingänge so verhalten, als seien sie auf 1 gelegt.

Ab nun werden für die Logikgatter stellvertretend folgende Symbole verwendet:

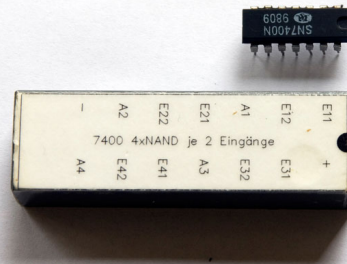


Das  $\geq$ -Zeichen beim OR-Gatter verdeutlicht, dass der Ausgang 1 wird, sobald die Summe der Eingangszustände  $A + B \geq 1$  sind. Beim XOR hingegen muss die Summe  $A + B$  genau 1 sein, symbolisiert durch  $= 1$ . Die 1 beim NOT bedeutet, dass der Eingangszustand einfach unverändert durchgeleitet wird. Die Invertierung geschieht durch die Rampe am Ausgang (ebenso beim NAND und NOR). Oft wird die Invertierung auch durch einen kleinen Kreis anstelle der Rampe symbolisiert. Die Nummer unter den Symbolen gibt die Bezeichnung der ICs an, die dieses logische Gatter beinhalten.

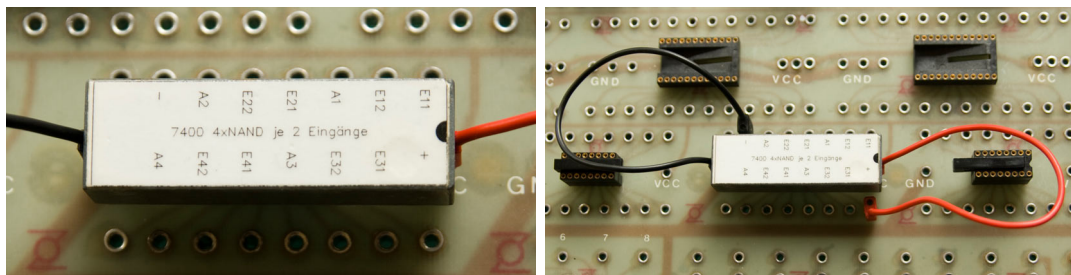
Zum Anschluss der ICs: Zuerst sucht man sich einen freien Sockel und schließt zur Stromversorgung je ein schwarzes/rotes Kabel links/rechts des Sockels bei Masse (GND)/+5 V (VCC) an:



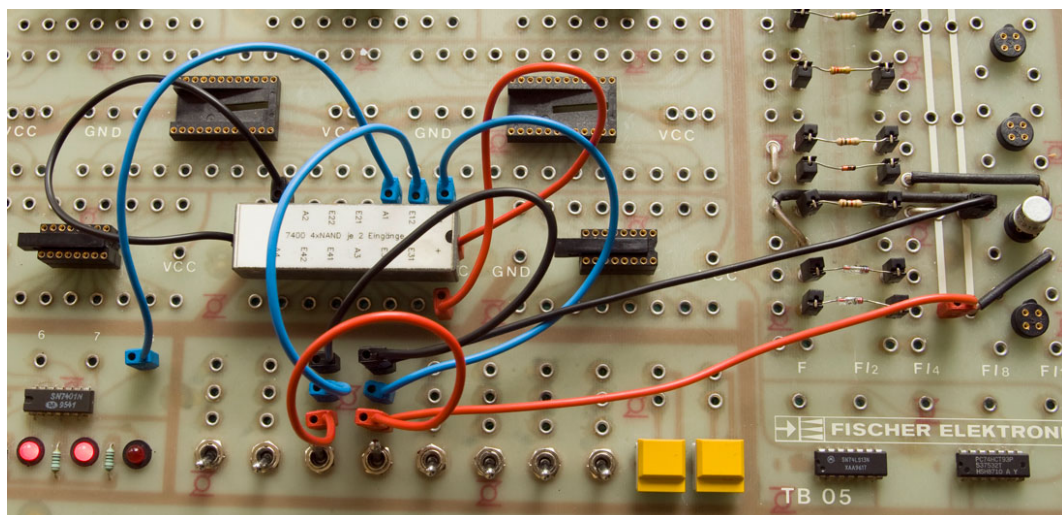
Nun sucht man sich das zu verwendende IC samt Anschlusskappe heraus (hier im Beispiel IC7400 - NAND).



Socket, IC sowie Anschlusskappe haben jeweils eine Kerbe (hier: schwarzer Halbkreis). Beim Anschließen ist darauf zu achten, dass diese immer bündig nach rechts zeigt. Das IC wird nun in den Socket eingesetzt und, falls alle Beinchen ordentlich in den Löchern des Sockels stecken, kräftig hineingedrückt. Anschließend kann die Anschlusskappe aufgesetzt werden. An dieser ist ersichtlich, wo die zuvor angebrachten Versorgungsleitungen eingesteckt werden müssen (GND in - sowie VCC in +).



Nun ist das IC betriebsbereit und kann verschaltet werden. In folgender Abbildung wurde ein *NAND* aufgebaut:



Über die beiden Schalter wird jeweils eine **1** in die Eingänge  $E_{11}$  und  $E_{12}$  eingegeben. Das Ergebnis (**0**) wird über den Ausgang  $A_1$  auf der Dioden ausgegeben.

## 2.1 Inverter (NOT-Gatter) aus NAND- oder NOR-Gatter

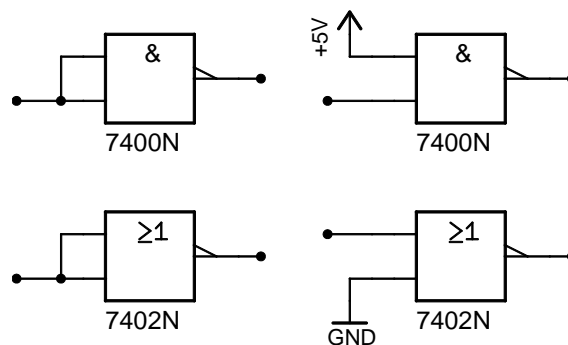
Nun soll aus einem NAND-/NOR-Gatter ein NOT-Gatter aufgebaut werden. In den Wahrheitstabellen der Funktionen betrachtet man dazu zunächst die Zeilen, in denen am Ausgang  $C$  die Negation des Eingangs  $A$  anliegt (mit Pfeil gekennzeichnet):

$A$	$B$	$C = \overline{A \wedge B}$	$A$	$B$	$C = \overline{A \vee B}$
→ 0	0	1	→ 0	0 <sub>*</sub>	1
→ 0	1 <sub>*</sub>	1	0	1	0
1	0	1	→ 1	0 <sub>*</sub>	0
→ 1	1 <sub>*</sub>	0	→ 1	1	0

Jetzt überlegt man, welche Bedingungen dann für  $B$  gelten müssen. Aus der ersten und vierten Zeile beider Wahrheitstabellen folgt  $B = A$ , d.h. die Eingangsleitungen des NAND-/NOR-Bausteins müssen verbunden werden.

Alternativ betrachtet man Zeile zwei und vier beim NAND (mit Sternchen gekennzeichnet): Hier liegt  $B$  jeweils auf **1**. Selbiges ergibt beim NOR für Zeile eins und drei:  $B$  liegt dauerhaft auf **0**.

Um einen Inverter zu erhalten können also zunächst bei beiden IC-Typen die Eingänge verbunden werden. Alternativ wird beim NAND der eine Eingang konstant auf **1** gelegt bzw. beim NOR einer der Eingänge konstant auf **0** gehalten:



## 2.2 XOR-Gatter

Das XOR (eXclusive OR) entspricht einem logischen „entweder oder“, ergibt also nur **1**, wenn genau einer der beiden Eingänge auf **1** liegt:

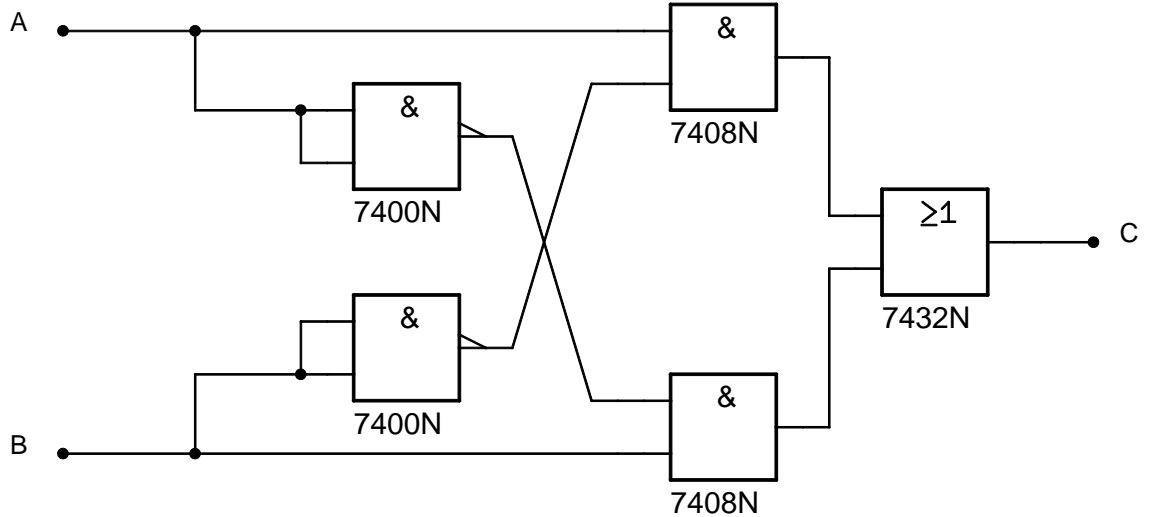
$A$	$B$	$C = A \underline{\vee} B$
0	0	0
0	1	1
1	0	1
1	1	0

Die disjunktive Normalform lässt sich direkt aus der Wahrheitstabelle ablesen, indem man die Zeilen betrachtet, in denen die Ausgangsvariable **1** ergibt, und die Bedingungen die jede dieser Zeile stellt (*UND*-Verknüpfungen) untereinander mit *ODER* verknüpft.

Im Falle des *XOR* ist die Ausgangsvariable  $C$  auf **1**, wenn der Eingang  $A = \mathbf{0}$  UND der Eingang  $B = \mathbf{1}$  ist, *ODER* wenn der Eingang  $A = \mathbf{1}$  UND der Eingang  $B = \mathbf{0}$  ist.

$$C = (\bar{A} \wedge B) \vee (A \wedge \bar{B})$$

Diese Formel kann nun aus 2\*NAND, 2\*AND und 1\*-OR realisiert werden:



### 2.3 XOR-Gatter aus NAND-Gatter

Die in Aufgabe 2.2 gewonnene Formel kann nun wie folgt in reine *NAND*-Schreibweise umgeformt werden:

$$C = (A \wedge \bar{B}) \vee (\bar{A} \wedge B) \quad (1)$$

$$= (A \wedge \bar{B}) \vee (A \wedge \bar{A}) \vee (\bar{A} \wedge B) \vee (B \wedge \bar{B}) \quad (2)$$

$$= A \wedge (\bar{A} \vee B) \vee B \wedge (\bar{A} \vee \bar{B}) \quad (3)$$

$$= A \wedge (\overline{\overline{\bar{A} \vee B}}) \vee B \wedge (\overline{\overline{\bar{A} \vee \bar{B}}}) \quad (4)$$

$$= A \wedge (\overline{\overline{A \wedge B}}) \vee B \wedge (\overline{\overline{A \wedge B}}) \quad (5)$$

$$= \underbrace{A \wedge (\overline{\overline{A \wedge B}})}_X \vee \underbrace{B \wedge (\overline{\overline{A \wedge B}})}_Y \quad (6)$$

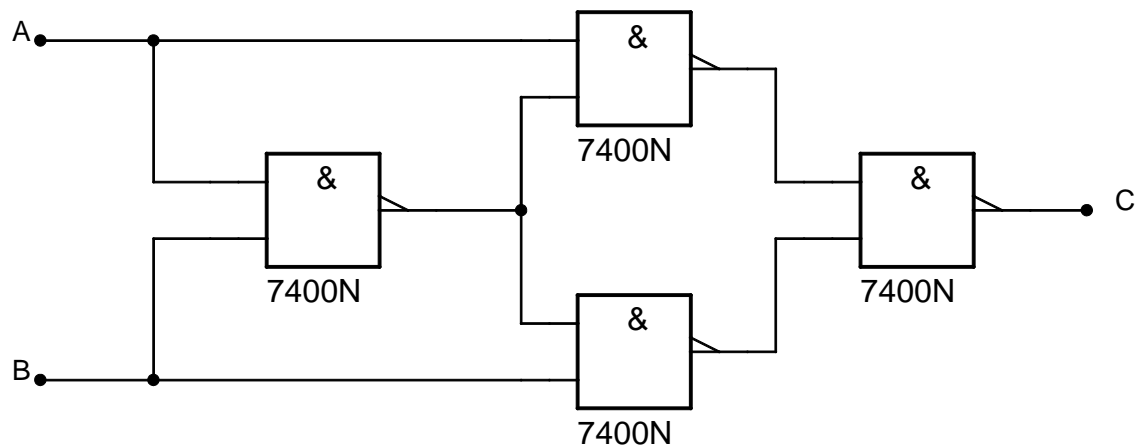
$$= \overline{\overline{A \wedge (\overline{\overline{A \wedge B}})} \wedge B \wedge (\overline{\overline{A \wedge B}})} \quad (7)$$

$$= \overline{\overline{A \wedge B} \wedge \overline{A \wedge B}} \quad (8)$$

Zunächst wird zweimal mit „ $\vee$ “ **0** erweitert (Gl. (2)), da die logische Verknüpfung  $(x \wedge \bar{x})$  eine Kontradiktion also immer **0** ist. In den ersten beiden Klammern aus Gl. (2) kann ein  $A$  ausgeklammert werden, da dieses in beiden Klammern vorkommt. Selbiges gilt auch für  $B$  in den letzten beiden Klammern (Gl. (3)). In Gl. (4) werden die beiden Klammern zweifach negiert was einer Äquivalenzumformung entspricht. Diese Umformung machen wir uns nun in Gl. (5) zu Nutze, indem nach dem De Morganschen Gesetz  $\overline{\overline{A \vee B}} = \overline{A} \wedge \overline{B}$

umgeformt wird. Jetzt wird die ganze Gleichung doppelt negiert (Gl. (6)) und wieder das De Morgansche Gesetz  $\overline{X \vee Y} = \overline{X} \wedge \overline{Y}$  angewendet, so dass letztendlich die in der Aufgabe geforderte Darstellung erreicht wird (Gl. (7) bzw. Gl. (8)).

Die so erhaltene Gleichung lässt sich nun aus lediglich 4 *NAND*-Gattern aufbauen:



Der Vorteil solcher Umformungen liegt darin, dass danach nur noch ein Gatterbausteintyp benötigt wird. Dies ist besonders für große Schaltungen von Bedeutung, da sich dadurch der Herstellungsaufwand sowie dessen Kosten extrem senken lassen.

### Fragen

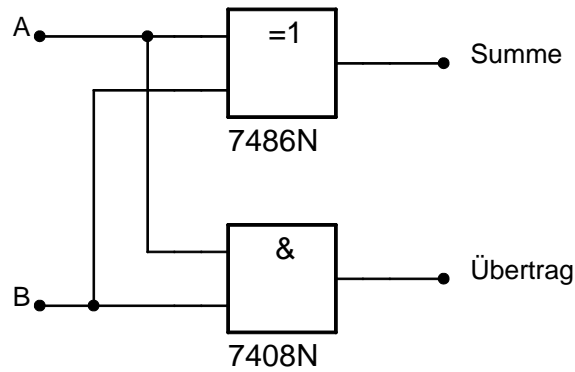
- Was bewirkt das Offenlassen eines Eingangs eines ICs?
- Wie erhält man aus einem NAND- oder NOR-Gatter ein NOT-Gatter?
- Wie kommt man, ausgehen von der Wahrheitstabelle einer Funktion auf deren disjunktive Normalform?
- Welche Vorteile ergeben sich aus einer Umformung der disjunktiven Normalform in die reine NAND- oder NOR-Schreibweise?

## 3 Addierer

### 3.1 Halbaddierer

A	B	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ein Halbaddierer addiert zwei 1-Bit-Binärzahlen und gibt das 2-Bit-Ergebnis in Form von Summe und Übertrag aus. Es lassen sich mit Halbaddierern keine größeren Rechnungen ausführen, da der Übertrag vorangehender Stufen nicht berücksichtigt werden kann.



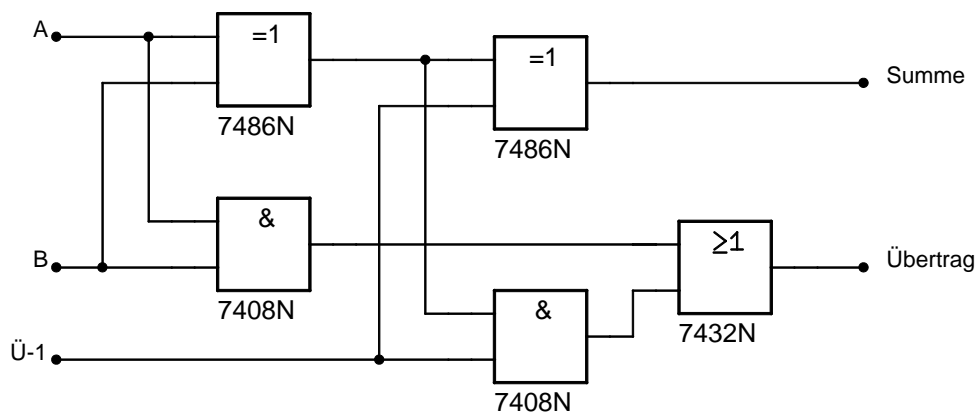
Die Summe ergibt sich, wie man aus der Wahrheitstafel leicht ablesen kann, durch Verwendung der *XOR* Verknüpfung. Es ist empfehlenswert das *XOR* als fertigen Baustein zu verwenden. Den Übertrag erhält man durch *UND*-Verknüpfung der beiden Eingänge.

Optimalerweise lässt man den Halbaddierer aufgebaut und erweitert ihn im nächsten Aufgabenteil zum Volladdierer.

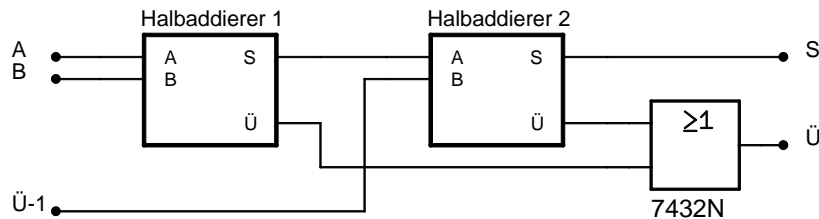
### 3.2 Volladdierer

A	B	C	S	Ü
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

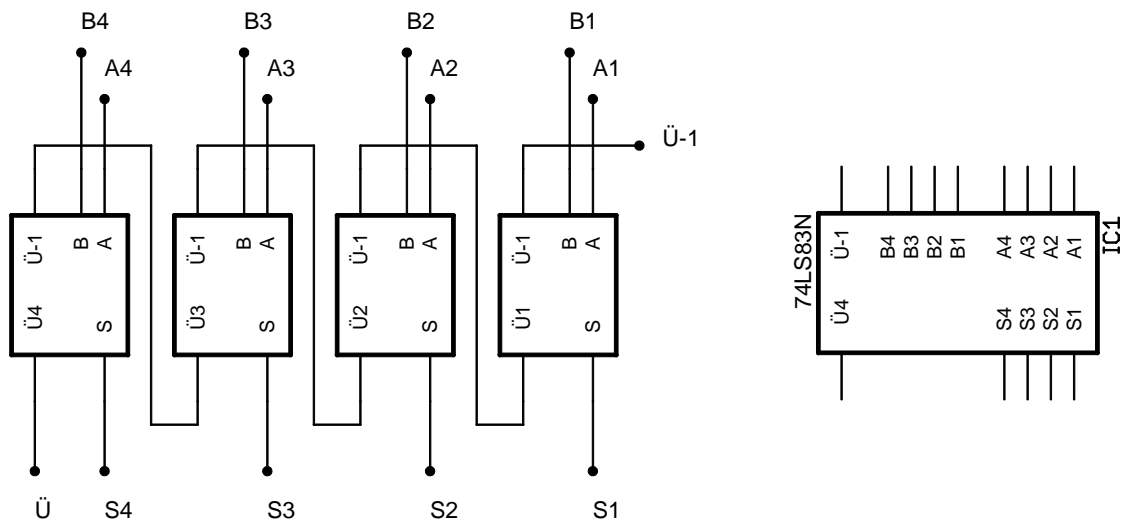
Um binäre Zahlen mit mehreren Bits zu addieren benötigt man Volladdierer, da beim Volladdierer auch der Übertrag der vorangehenden Stufe berücksichtigt werden kann. Ein Volladdierer lässt sich aus zwei Halbaddierern aufbauen und kann drei 1-Bit-Binärzahlen addieren. Der dritte Eingang  $\ddot{U}_{-1}$  aus folgendem Schaubild entspricht der Spalte *C* aus der Wahrheitstabelle von oben. Da alle drei Eingänge (A,B,C) völlig gleichwertig sind, wurde dort zunächst keine Unterscheidung vorgenommen.



Wir bauen zusätzlich zum im vorangehenden Aufgabenteil bereits aufgebauten Halbaddierer einen weiteren Halbaddierer auf und geben auf dessen Eingänge den Summe-Ausgang des „alten“ Halbaddierers sowie die dritte 1-Bit-Zahl (= der Übertrag der vorangehenden Stufe). Der Ausgang dieses „neuen“ Halbaddierers ist die Summe des Volladdierers. Zusätzlich Verknüpfen wir noch die Übertragsausgänge beider Halbaddierer mit *ODER*, dies ergibt den Übertragsausgang des Volladdierers.



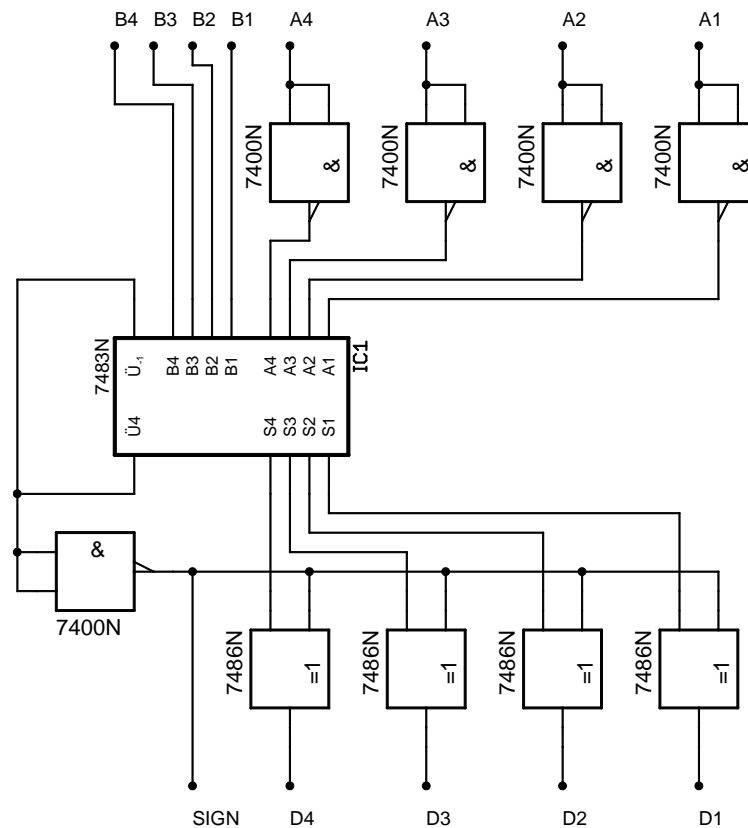
Zur Addition größerer Zahlen können nun mehrerer solcher Volladdierer zusammengeschlossen werden. Als Beispiel wurde hier ein 4-Bit-Volladdierer gewählt, der in der nächsten Aufgabe in Form eines ICs Verwendung findet:



Der erste Volladdierer ganz rechts addiert die beiden Stellen  $A_1, B_1$  der 4-Bit-Zahlen  $A_4A_3A_2A_1, B_4B_3B_2B_1$  (wir vernachlässigen zunächst den Eingang  $\ddot{U}_{-1}$ ). Die Summe  $A_1 + B_1$  wird an  $S_1$  ausgegeben. Ein eventuell auftretender Übertrag aus dieser Rechnung wird über  $\ddot{U}_1$  in den Übertragseingang  $\ddot{U}_{-1}$  des zweiten (von rechts) Volladdierers gegeben. Dieser führt nun, unter Berücksichtigung des Übertrags aus der ersten Stufe, die Rechnung  $A_2 + B_2 + \ddot{U}_1$  aus und gibt deren Summe an  $S_2$  aus. Der eventuell auftretende Übertrag  $\ddot{U}_2$  wird wieder an die nächste Stufe weitergegeben. So führt sich das Ganze fort bis zum letzten Volladdierer (ganz links). Der Übertrag  $\ddot{U}_4$  aus der letzten Rechnung ( $A_4 + B_4 + \ddot{U}_3$ ) schließlich wird nach außen geführt und kann so optional von einem weiteren Addierer verarbeitet werden. Genauso kann über den  $\ddot{U}_{-1}$ -Eingang rechts oben ein Übertrag aus einer vorangehenden Rechnung eingegeben werden.

### 3.3 Subtrahierer

Der 4-Bit Subtrahierer lässt durch geschickte Verschaltung aus einem 4-Bit-Addierer, 5 *NOT*-Gattern und 4 *XOR*-Gattern aufbauen.



Die abzuziehende Zahl  $A$  mit den binären Komponenten  $A_4A_3A_2A_1$  wird invertiert, indem die einzelnen Komponenten jeweils durch ein zu einem *NOT* verschalteten *NAND*-Gatter negiert werden. Alternativ kann dies auch durch Vertauschen der Versorgungsleitungen der Schalter zur Eingabe der Zahl  $A$  erfolgen. Die Negation entspricht folgender Rechenoperation:

$$\bar{A} = 15 - A$$

Der 4-Bit Volladdierer rechnet damit:

$$\begin{aligned} S &= B + \bar{A} + \ddot{U}_{-1} \\ &= B - A + 15 + \ddot{U}_{-1} \end{aligned}$$

Da der Übertragseingang  $\ddot{U}_{-1}$  an den Übertragsausgang  $\ddot{U}_4$  angeschlossen ist, ist an dieser Stelle eine Fallunterscheidung notwendig. Falls im Volladdierer ein Übertrag entsteht, so wird dieser auch zu den Eingangszahlen hinzuaddiert. Wir müssen daher unterscheiden, ob  $B - A < 0$ ,  $B - A = 0$  oder  $B - A > 0$  vorliegt:

**Fall 1:**  $B - A > 0$ : In diesem Fall ist die Summe des 4-Bit Volladdierers größer als seine maximal darstellbare Zahl und es wird am Übertragsausgang  $\ddot{U}_4$  ein entstandener

Übertrag (1) ausgegeben. Da dieser Übertrag jedoch an den Übertragseingang angeschlossen ist, wird durch  $\ddot{U}_{-1} = \mathbf{1}$  noch eine 1 addiert und die Rechnung wird zu:

$$\begin{aligned} S &= B + \bar{A} + \ddot{U}_{-1} \\ &= B - A + 15 + 1 \\ &= B - A + 16 \end{aligned}$$

Der Übertragsausgang  $\ddot{U}_4$  ist  $\mathbf{1}$ , somit ist das Vorzeichenbit  $SIGN$ , welches mit einem  $NAND$ -Gatter als Negation von  $\ddot{U}_4$  gebildet wird  $\mathbf{0}$ , was einem  $+$  entspricht. Die  $XOR$ -Verknüpfungen, welche die Summenausgänge des 4-Bit-Addierers  $S_4S_3S_2S_1$  mit den Ausgabebits  $D_4D_3D_2D_1$  verknüpfen, verändern das Signal nicht, da an jedem dieser  $XOR$ -Gatter an einem Eingang  $\mathbf{0}$  anliegt ( $Z \vee 0 = Z$ ). Da es nur vier Ausgangsbits  $D_4D_3D_2D_1$  gibt, werden Zahlen größer als 15 (der größten mit vier Bits darstellbaren Zahl) abgeschnitten

$$D = S \bmod 16$$

Wenn wir beachten, dass es sich um zwei 4-Bit Eingangsvariablen handelt, welche voneinander abgezogen werden, so dass maximal einfaches Überschreiten der 16 möglich ist, folgt für  $D$ :

$$\begin{aligned} D &= S - 16 \\ &= (B - A + 16) - 16 \\ &= B - A \end{aligned}$$

**Fall 2:**  $B - A < 0$ : Nun hingegen ist die Summe des 4-Bit Volladdierers kleiner als die maximal darstellbare Zahl, es entsteht daher kein Übertrag,  $\ddot{U}_4$  ist somit  $\mathbf{0}$ . Da der Übertragsausgang an den Übertragseingang angeschlossen ist, ist daher auch  $\ddot{U}_{-1} \mathbf{0}$ . Die Rechnung wird zu:

$$\begin{aligned} S &= B + \bar{A} + 0 \\ &= B - A + 15 \end{aligned}$$

Das Vorzeichenbit  $SIGN$  ist damit, als Negation von  $\ddot{U}_4$ ,  $\mathbf{1}$  (entspricht  $-$ ). Da  $Z \vee 1 = \bar{Z}$  invertiert nun die  $XOR$ -Verknüpfung, welche die Ausgangsbits  $D_4D_3D_2D_1$  aus den Summenausgängen des Addierers  $S_4S_3S_2S_1$  bildet. Umschreibt man die Negierung wie oben, so gilt:

$$\begin{aligned} D &= \bar{S} \\ &= 15 - S \\ &= 15 - (B - A + 15) \\ &= A - B \end{aligned}$$

Beachtet man nun noch, dass das Vorzeichenbit gesetzt ist:

$$\begin{aligned} D &= (-1) * (A - B) \\ &= B - A \end{aligned}$$

**Fall 3:**  $B - A = 0$  (**Sonderfall**) In diesem Fall ist das Ergebnis davon abhängig, ob in der vorangehenden Rechnung  $\ddot{U}_{-1}$  **0** oder **1** gewesen ist.

War  $\ddot{U}_4 = \ddot{U}_{-1} = 1$  und lag damit Fall 1 vor, so verändert sich die Rechnung des Addierers zu:

$$\begin{aligned} S &= B - A + 15 + \ddot{U}_{-1} \\ &= \underbrace{B - A}_{=0 \text{ n.V.}} + 15 + 1 \\ &= 16 \end{aligned}$$

Da  $S = 16$  nur mit Übertrag darstellbar ist, bleibt der Übertragsausgang  $\ddot{U}_4 = \ddot{U}_{-1} = 1$ , somit wird das Vorzeichenbit *SIGN* durch die Negation **0** und die *XOR*-Verknüpfungen verändern die Summenausgänge nicht ( $Z \vee 0 = Z$ ). Beim Bilden des Übertrags findet analog zu Fall 1 eine Beschränkung auf modulo 16 statt.

$$\begin{aligned} D &= S \bmod 16 \\ &= 16 \bmod 16 \\ &= 0 \end{aligned}$$

Das Vorzeichenbit *SIGN* ist **0**, das Ergebnis ist folglich  $D = +0$ . Lag im vorangehenden Schritt jedoch Fall 2 vor ( $\ddot{U} = \ddot{U}_{-1} = 0$ ), so ergibt die Summe des 4-Bit-Addierers:

$$\begin{aligned} S &= B - A + 15 + \ddot{U}_{-1} \\ &= \underbrace{B - A}_{=0 \text{ n.V.}} + 15 + 0 \\ &= 15 \end{aligned}$$

Nun geht es analog zu Fall 2 weiter, da  $S = 15$  vom 4-Bit-Addierer ohne Übertrag darstellbar ist, bleibt  $\ddot{U}_4$  und damit auch  $\ddot{U}_{-1}$  auf **0**. Durch die Negation ist das Vorzeichenbit *SIGN* damit **1**. Wegen  $Z \vee 1 = \overline{Z}$  invertieren die *XOR*-Verknüpfungen die Summenausgänge  $S_4 S_3 S_2 S_1$ .

$$\begin{aligned} D &= \overline{S} \\ &= 15 - S \\ &= 15 - 15 \\ &= 0 \end{aligned}$$

Das Vorzeichenbit *SIGN* ist **1**, das Ergebnis ist somit  $D = -0$ , da aber  $+0 = -0$ , erhält man bei Fall 3 bei beiden möglichen Anfangszuständen identische Ergebnisse. Lediglich der Rechenweg weicht bei den unterschiedlichen Anfangszuständen voneinander ab.

Dieser 3. Fall nimmt eine Sonderstellung ein, wichtig für das Verständnis des Subtrahierers sind eigentlich nur die beiden Fälle 1 und 2, Fall 3 ist jedoch der Vollständigkeit halber erwähnt.

**Beispiel 1:**  $B = 5$  und  $A = 8$ , dann ist  $B - A + 15 = 5 - 8 + 15 = 12 < 16$ . Es liegt Fall 2 vor.  $S = B - A + 15 + 0 = 12$  und  $D = 15 - S = 15 - 12 = 3$ . Das Vorzeichenbit ist 1, daher ist das Ergebnis als  $-3$  zu interpretieren.

**Beispiel 2:**  $B = 12$  und  $A = 7$ , dann ist  $B - A + 15 = 12 - 7 + 15 = 20 \geq 16$ . Es liegt Fall 1 vor.  $S = B - A + 15 + 1 = 21$  und  $D = S \bmod 16 = 21 \bmod 16 \equiv 21 - 16 = 5$ . Das Vorzeichenbit ist 0, daher ist das Ergebnis 5.

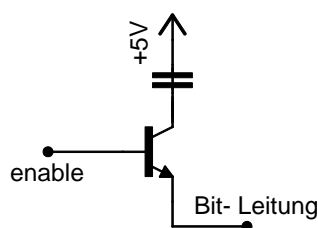
### Fragen

- Welches Problem ergibt sich beim Halbaddierer?
- Wie kommt man darauf, aus welchen Gattern der Halbaddierers aufgebaut ist?
- Wozu wird das ODER beim Volladdierer benötigt?
- Wozu werden beim Subtrahierer die XORs benötigt und in welchem Fall haben sie keine Wirkung?
- Wieso wird der Übertragsausgang auf den Übertragseingang rückgekoppelt und wann gibt dieser einen Übertrag aus?
- Welches Problem besteht beim Subtrahierer im Bezug auf die Ausgabe der Null?

## 4 Speicherelemente

Ein Flip-Flop ist eine einfach zu realisierende und schnelle Speicherform. Ein binärer Zustand kann über den Eingang eingegeben werden und wird daraufhin vom Flip-Flop so lange gehalten ( $\Rightarrow$  Speicher), bis ein neuer Zustand gesetzt wird.

**Hinweis:** Im RAM eines Computers finden keine Flip-Flops Verwendung, da sonst für jedes Bit mindestens ein komplettes Reset-Set-Flip-Flop benötigt werden würde, hierbei ist der Aufwand an Transistoren sehr groß. Stattdessen wird ein dynamischer Speicher ( $\rightarrow$  D-RAM) bestehend aus einem Kondensator und einem Transistor pro Bit eingesetzt:

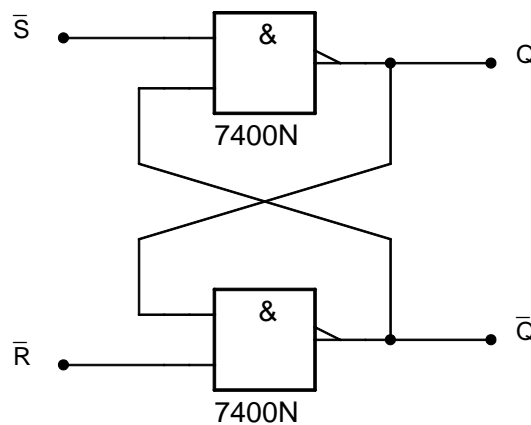


Die Geschwindigkeit dieser Art von Speicher ist zwar geringer als mit Flip-Flops, dafür ist aber auch der Aufwand viel niedriger. Der CPU-Cache hingegen läuft bei aktuellen CPUs mit voller CPU-Geschwindigkeit, daher müssen Taktfrequenzen im mehrere GHz-Bereich erreicht werden, was sich nur durch solche aufwendigen FFs realisieren lässt.

#### 4.1 RS-Flip-Flop (RS-FF)

$S$	$R$	$Q_n$	$\overline{Q}_n$	
0	0	$Q_{n-1}$	$\overline{Q}_{n-1}$	keine Änderung (speichern)
0	1	0	1	setze <b>0</b>
1	0	1	0	setze <b>1</b>
1	1	1	1	$Q = \overline{Q}$ : verboten

Das **Reset-Set-Flip-Flop** (kurz: RS-FF) ist das einfachste Flip-Flop. Es verfügt über zwei getrennte Eingänge **Set** und **Reset**. Im Schaltbild wurden negierte Set- und Reset-Eingänge verwendet damit wir die gleiche Situation wie beim noch zu besprechenden, taktzustandsgesteuerten RST-FF haben, bei dem ja durch die vorgeschalteten *NAND*-Gatter der Taktstufe die Eingänge des FFs negiert werden. Somit ergibt sich die gleiche Wahrheitstabelle für die beiden Flip-Flops. Die ausführliche Diskussion der Funktionsweise wird nur am RS-FF durchgeführt und ist durch diesen „Trick“ beim RST-FF identisch.



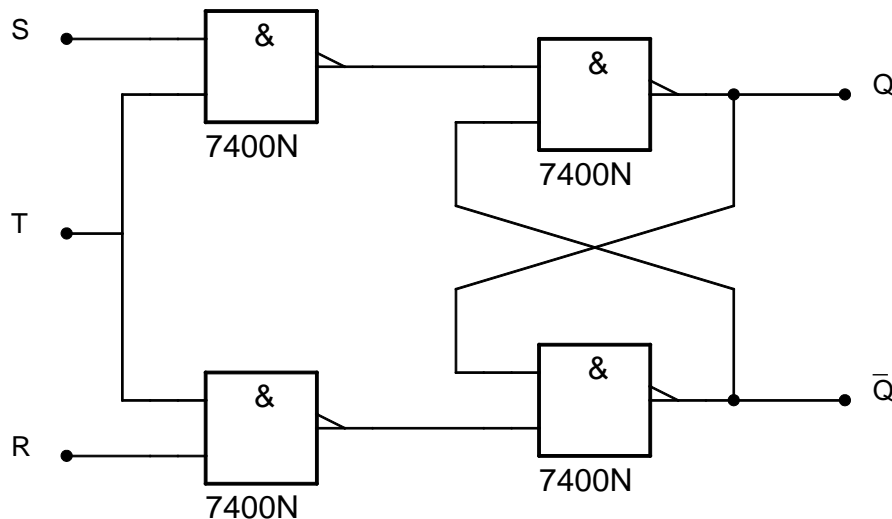
Wird an Set **1** angelegt, so ist  $\overline{S} = 0$  und damit ergibt die obere *NAND*-Verknüpfung, egal was an ihrem anderen Eingang anliegt, wegen  $A \wedge 0 = 1$ , sprich es wird der Ausgang  $Q$  auf **1** gesetzt. Analog wird, wenn an Reset **1** angelegt wird, und damit  $\overline{R} = 0$  ist, die untere *NAND*-Verknüpfung (also der inverse Ausgang  $\overline{Q}$ ) **1**. Jetzt erkennen wir auch direkt das Problem dieser Schaltung: Wird sowohl an Set wie an Reset **1** angelegt, so ergeben beide *NAND*-Verknüpfungen **1**. In diesem Falle wäre  $Q = \overline{Q} = 1$ , was als falsche Aussage gilt, da nun  $\overline{Q}$  nicht mehr das negierte zu  $Q$  ist. Dies ist der sogenannte „Verbotene Zustand“ des Flip-Flops. Legt man jedoch an Set und an Reset **0** an, also  $\overline{S} = \overline{R} = 1$ , so liegt bei beiden *NAND*-Gatter an einem der Eingänge jeweils **1**. Da  $A \wedge 1 = \overline{A}$  ist der Ausgangszustand  $Q$  bzw.  $\overline{Q}$  abhängig vom jeweilig anderen Eingang des Gatters. Da diese Eingänge „überkreuzt“ an den Ausgänge des jeweilig anderen Gatters angeschlossen sind, ergibt die obere *NAND*-Verknüpfung  $Q = \overline{Q} \wedge 1 = \overline{Q} = Q$  und analog  $\overline{Q} = Q \wedge 1$ . Folglich bleiben bei dieser Belegung die Ausgangszustände  $Q$  und  $\overline{Q}$  erhalten.

Es bietet sich an das RS-FF auf dem Sockel ganz rechts unten nahe den Schaltern aufzubauen, da diese Schaltung später zu einer Entprellschaltung erweitert und in den folgenden Aufgabenteilen weiterverwendet wird.

## 4.2 Getaktetes RS-Flip-Flop (RST-FF)

$T$	$S$	$R$	$Q_{n+1}$	$\overline{Q}_{n+1}$	
0	0	0	$Q_n$	$\overline{Q}_n$	kein Takt $\Rightarrow$ keine Änderung
0	0	1	$Q_n$	$\overline{Q}_n$	kein Takt $\Rightarrow$ keine Änderung
0	1	0	$Q_n$	$\overline{Q}_n$	kein Takt $\Rightarrow$ keine Änderung
0	1	1	$Q_n$	$\overline{Q}_n$	kein Takt $\Rightarrow$ keine Änderung
1	0	0	$Q_n$	$\overline{Q}_n$	keine Änderung (speichern)
1	0	1	0	1	setze <b>0</b>
1	1	0	1	0	setze <b>1</b>
1	1	1	1	1	$Q = \overline{Q}$ : verboten

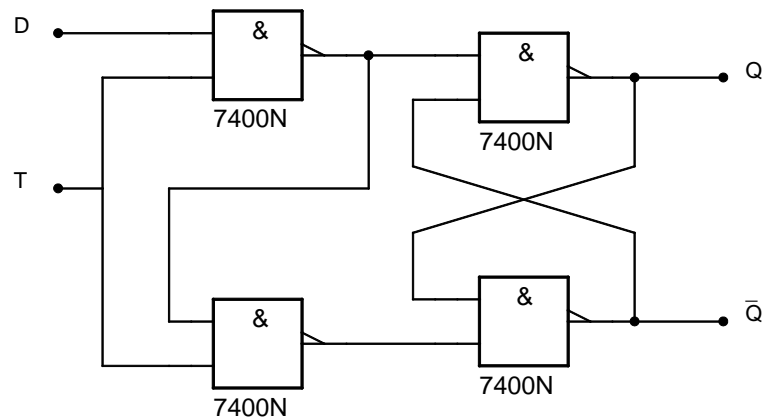
Ein getaktetes Reset-Set-Flip-Flop (kurz: RST-Flip-Flop) ist um einen zusätzlichen Eingang, den Takteingang, erweitert. Die beiden anderen Eingänge, Set und Reset, sind jeweils mit dem Takteingang über *NAND* verknüpft, so dass, nur wenn am Takteingang **1** anliegt, der Schaltzustand des Flip-Flops überhaupt beeinflusst werden kann, d.h. der Takt „enabled“ die Eingänge. Dieses Flip-Flop ist damit taktzustandsgesteuert.



Die Funktionsweise ist sonst identisch zu RS-FF (hier kommt der Vorteil der inversen Set- und Reset-Eingänge des RS-FF zu tragen).

## Data-Flip-Flop (D-FF)

Wird beim RST-FF der Reset Eingang über eine Negation an den Set-Eingang angeschlossen (hierzu kann man das *NAND*-Gatter des Set Einganges der Taktstufe verwenden), so ergibt sich ein neuer Eingang, der sog. Data-Eingang  $D$ . Liegt der Taktzustand **1** an und legt man an diesen Eingang ( $D$ ) **1** an, so wird  $Q$  auf **1** gesetzt. Dies wird solange gehalten bis man gleichzeitig  $T = 1$  und  $D = 0$  anlegt. Das D-FF speichert folglich die am  $D$ -Eingang angelegten Informationen bis zum nächsten Taktzustand **1**. Dieses Flip-Flop hat keinen verbotenen Zustand da immer  $\overline{S} \neq \overline{R}$  gewährleistet ist.

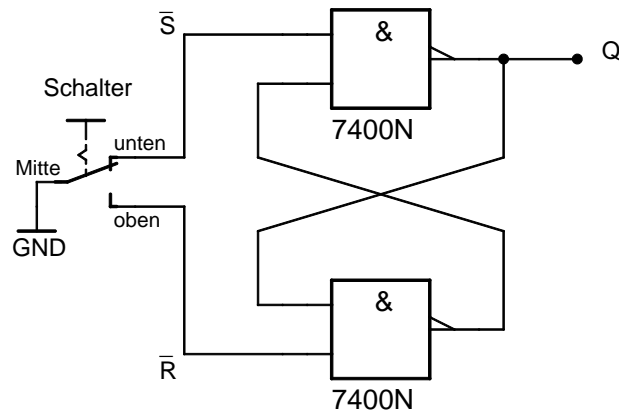


### Entprellen eines Schalters

Bei Schaltern handelt es sich meistens um mechanische Bauteile, bei denen je nach Schalterstellung Kontakt hergestellt oder der Kontakt entfernt ist. Beim Einschalten, also dem Umschalten von *kein Kontakt*  $\rightarrow$  *Kontakt*, kann durch elastisches Verhalten der sich bewegenden Teile im Schalter es zu einem „Abprallen“ kommen. So wird der Kontakt hergestellt, danach aber, unter Umständen sogar mehrmals, kurzzeitig wieder unterbrochen und erneut hergestellt. Wird ein von einem solchen Schalter erzeugtes Signal als Taktsignal für beliebige taktf flankengesteuerte Schaltungen (Bsp.: Schieberegister, Zähler, ...) verwendet, so kann bei einem einzigen Umschalten des Schalters die taktf flankengesteuerte Schaltung mehrere Taktf flanken registrieren. Durch die hohe Geschwindigkeit der Flip-Flops reichen diese durch das Abprallen des bewegenden Schaltelements erzeugten Signaländerungen aus, als erneute Signalfanke registriert zu werden und damit ein unbeabsichtigtes Schieben (beim Schieberegister) oder Hochzählen (beim Zähler) auszulösen.

Dies lässt sich umgehen, wenn als Taktsignal immer das Signal des Taktgenerators des Boards verwendet wird, am besten den langsamsten Ausgang  $F/16$  da die anderen Taktsignale zu schnell für saubere Beobachtungen sind.

Um jedoch trotzdem Taktsignale mit den Schaltern erzeugen zu können, lässt sich relativ einfach eine „Entprellschaltung“ aufbauen, welche aus einem RS-FF besteht (das im letzten oder vorletzten Aufgabenteil aufgebaute FF verwenden). Es wird hierzu an den mittleren Anschluss eines Schalters (der Steckplatz, welcher sonst als Ausgang verwendet wird) 0 angelegt. Nun kann der untere Ausgang mit Set  $\bar{S}$  und der obere mit Reset  $\bar{R}$  eines RS-FF verbunden werden. Am Ausgang  $Q$  dieses FF kann nun das entprellte Signal abgegriffen werden.

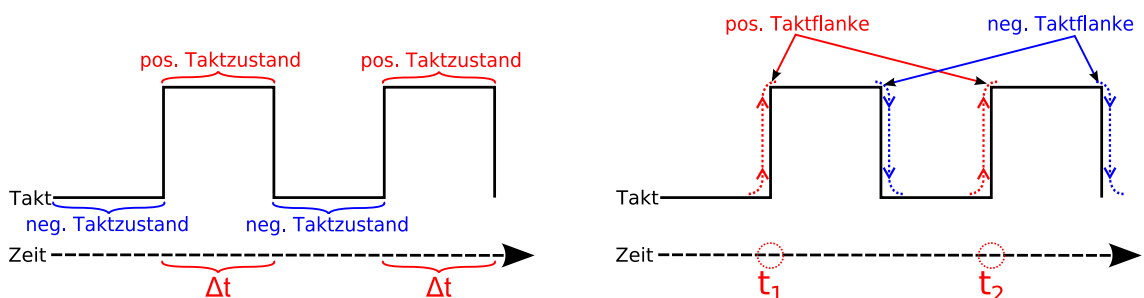


Diese Schaltung macht sich den schon mehrmals erwähnten Umstand zu Nutze, dass an einem „leeren“, also nicht angeschlossenen Eingang logisch **1** anliegt. Die im Board eingebauten Schalter sind „Umschalter“, welche also ihren „Ausgang“ (mittlerer Anschluss) entweder mit dem oberen oder unteren Anschluss verbinden. Da diesmal der Schalter jedoch anders herum verwendet wird, liegt also entweder am oberen oder am unteren Anschluss **0** an und der jeweils andere Anschluss ist offen. Kommt es beim Umschalten nun zum „Prellen“, so wird kurzzeitig mehrmals die Verbindung zum mittleren Anschluss unterbrochen und danach sofort wiederhergestellt, es kann jedoch, falls wir gerade auf „oben“ geschaltet haben nicht vorkommen dass plötzlich an „unten“ ebenfalls **0** anliegt. Dies ist nicht möglich da dies die entgegengesetzte Schalterstellung benötigen würde. In unserer Entprellschaltung wird also je nach Schalterstellung entweder an  $\bar{S}$  oder an  $\bar{R}$  **0** angelegt und damit der Ausgang  $Q$  des FF auf **1** bzw. **0** gesetzt. Durch das Prellen wird nun aber nicht der entgegengesetzte Zustand sondern nur der Speicherzustand, also  $\bar{S} = \bar{R} = 1$  angelegt, dieser verändert aber den Ausgangszustand des FF nicht (siehe RS-FF). Somit ist das Prellen kompensiert.

Diese Schaltung wird aus dem RS- oder RST-FF aufgebaut und für das Taktsignal der nachfolgenden Schaltungen verwendet.

## Unterschiede Taktzustands- und Taktflankensteuerung

Es gibt zwei unterschiedliche Arten von Taktsteuerung bei Flip-Flops.



**Taktzustandsgesteuerte FFs (linke Abbildung):** Bei Flip-Flops dieser Kategorie (z.B. RST-FF, D-FF) können die Ausgänge solange über die Eingänge manipuliert wer-

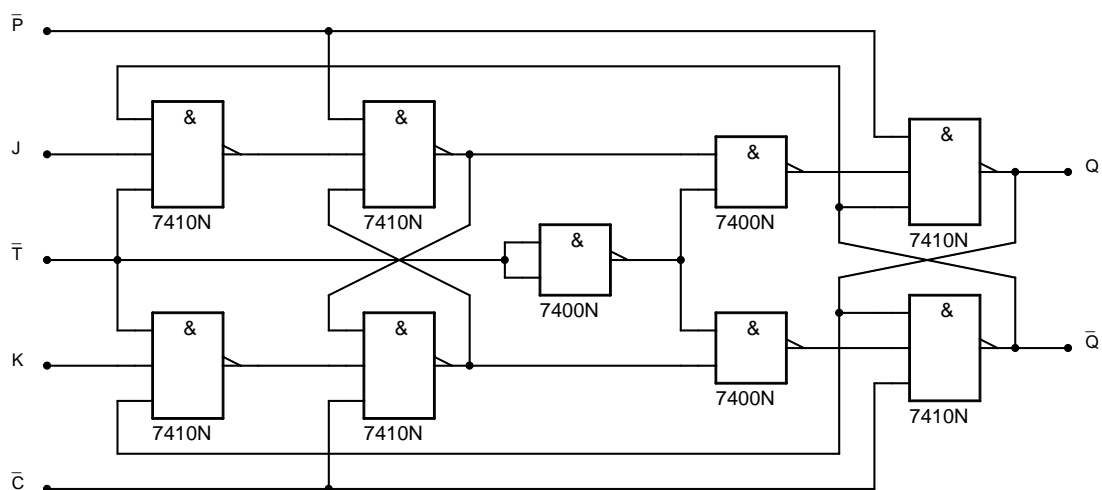
den, wie der zugehörige Taktzustand vorliegt. Bei einem positiv taktzustandsgesteuerten FF kann, solange das Taktsignal **1** ist (also im gesamten Zeitintervall  $\Delta t$ ) der Ausgangszustand durch die Eingänge verändert werden. Falls man die Eingänge in  $\Delta t$  mehrfach hin- und herschaltet, wird dies auch instantan auf die Ausgänge übertragen.

**Taktflankengesteuerte FFs (rechte Abbildung):** Bei diesen wird der Übergang des Taktes von einem Zustand zum anderen benötigt um die Ausgänge durch die Eingänge zu beeinflussen. Beim positiv taktflankengesteuerten FF wird zu dem Zeitpunkt, in dem der Takt von **0** auf **1** übergeht, der Ausgangszustand durch die momentane Belegung der Eingänge bestimmt. Die Beeinflussung des Ausganges durch die Eingänge findet in einem einzigen Zeitpunkt  $t_n$  und nicht wie bei Taktzustandssteuerung in einem ganzen Zeitintervall statt. Hierdurch zeigt mehrfaches Hin- und Herschalten der Eingänge keine Wirkung. Es ist nur ihre momentane Belegung zum Zeitpunkt  $t_n$  von Relevanz.

### 4.3 Jump-Kill-Master-Slave-Flip-Flop (JK-MS-FF)

Das **Jump-Kill-Master-Slave-Flip-Flop** ist ein taktflankengesteuertes Flip-Flop. Nicht der aktuelle Zustand des Taktsignals (wie beim RST-Flip-Flop) verarbeitet die Informationen, sondern der Übergang des Taktes, also die Taktflanke. In der Wahrheitstafel bedeutet  $+$  positive Taktflanke, also der Übergang des Taktes vom Zustand **0** in den Zustand **1**, dementsprechend steht  $-$  für die negative Taktflanke, also für den Übergang von **1** auf **0**. Merke: Eine Taktflanke ist immer ein Übergang des Taktes von einem Zustand in einen anderen, also die „Ableitung“ des Taktsignals.

Ein JK-MS-FF besteht aus zwei hintereinandergeschalteten RST-FFs, vorne dem sog. Master und dahinter dem Slave. Zwischen den beiden RST-FFs ist die Taktleitung durch ein *NAND*-Gatter negiert. Somit sind Slave und Master immer bei entgegengesetzten Taktzuständen „enabled“, also nie beide gleichzeitig. Dies führt insgesamt zur Taktflankensteuerung des gesamten Flip-Flops.



Bei negativem Taktzustand, also wenn an  $\bar{T}$  **1** anliegt (der Takteingang des Flip-Flops ist negiert), ist das vordere (Master-)RST-FF „enabled“. Jetzt kann durch  $J$  oder  $K$

der Speicherzustand dieses FF verändert werden. Wie beim normalen RST-FF ergibt  $J = 0 \ K = 0$  keine Änderung der Ausgänge des Masters,  $J = 1 \ K = 0$  setzt  $Q_{\text{Master}}$  auf **1** und  $J = 0 \ K = 1$  auf **0**. Anders als beim RST-FF gibt es keinen verbotenen Zustand, das Anlegen von  $J = 1 \ K = 1$  schreibt durch die kreuzweise Rückkopplung auf die Ausgänge des Slave-FF den Ausgangszustand des Slave-FF invertiert in den Master  $Q_{\text{Master}} = \overline{Q_{\text{Slave}}}$ .

In diesem Taktzustand ( $\overline{T} = 1$ ) ist der Slave gesperrt und kann nicht verändert werden.

Wechselt der Taktzustand und ist nun positiv also  $\overline{T} = 0$ , so ist der Master gesperrt und kann nicht verändert werden, das hintere RST-FF jedoch ist nun „enabled“ und verarbeitet die an ihm anliegenden Informationen, also die Ausgangszustände des Masters. Liegt am Ausgang des Masters  $Q_{\text{Master}}$ , welcher an den Set-Eingang des Slave angeschlossen ist, **1** an, so wird in den Ausgang des Slaves  $Q_{\text{Slave}}$  **1** geschrieben. Analog wird bei  $Q_{\text{Master}} = 0$  **0** geschrieben. Dies sind die einzigen beiden Möglichkeiten, da  $Q_{\text{Master}} \neq \overline{Q_{\text{Master}}}$  ( $\rightarrow$  kein Speicherzustand oder verbotener Zustand).

Um Informationen die an den Jump- und Kill-Eingängen des JK-MS-FF anliegen zu verarbeiten, benötigt man zuerst den negativen Taktzustand  $\overline{T} = 1$  um diese in den Master zu schicken. Danach wird aber noch der positive Taktzustand  $\overline{T} = 0$  benötigt um die Informationen auch vom Slave verarbeiten zu lassen. Erst nach beiden Taktzuständen in dieser Reihenfolge können die Informationen von ganz vorne ( $J, K$ ) zu den Ausgängen ( $Q, \overline{Q}$ ) des JK-MS-FF gelangen. Betrachtet man nun wieder das JK-MS-FF als Ganzes reicht also nicht mehr das Anliegen eines bestimmten Taktzustandes, statt dessen benötigen wir beide Taktzustände in festgelegter Reihenfolge, erst negativ  $\overline{T} = 1$  und danach positiv  $\overline{T} = 0$ . Dies entspricht wie oben festgestellt der positiven Taktflanke. Dieser Aspekt, dass für die Verarbeitung der Informationen beim JK-MS-FF erst der negative und danach der positive Taktzustand benötigt wird, lässt uns von einem positiv taktflankengesteuerten Flip-Flop sprechen. Ohne die zwischen den beiden RST-FFs in der Taktleitung eingebaute *NOT*-Verknüpfung würden beide Stufen (Master sowie Slave) auf den gleichen Taktzustand reagieren und die Ein- und Ausgänge wären nicht entkoppelt.

$J$	$K$	Taktflanke	$Q_{n+1}$	$\overline{Q}_{n+1}$	
0	0	+	$Q_n$	$\overline{Q}_n$	keine Änderung (speichern)
0	0	–	$Q_n$	$\overline{Q}_n$	Falsche Taktflanke $\Rightarrow$ keine Änderung
0	1	+	0	1	setze <b>0</b>
0	1	–	$Q_n$	$\overline{Q}_n$	Falsche Taktflanke $\Rightarrow$ keine Änderung
1	0	+	1	0	setze <b>1</b>
1	0	–	$Q_n$	$\overline{Q}_n$	Falsche Taktflanke $\Rightarrow$ keine Änderung
1	1	+	$\overline{Q}_n$	$Q_n$	toggeln
1	1	–	$Q_n$	$\overline{Q}_n$	Falsche Taktflanke $\Rightarrow$ keine Änderung

Bei den bisher nicht angesprochenen Eingängen  $\overline{\text{Preset}}$  und  $\overline{\text{Clear}}$  handelt es sich um parallele Eingänge, mit welchen man den Ausgangszustand direkt und damit völlig unabhängig vom Takt beeinflussen kann. Damit „normales“ Schalten möglich ist, müssen sowohl  $\overline{P}$  als auch  $\overline{C}$  auf **1** liegen, da sonst der Ausgang immer direkt festgelegt ist. Dies ist z.B. gewährleistet, wenn man diese Eingänge „offen“ lässt (wie schon erwähnt: Offene Eingänge gelten als mit **1** belegt).

**Hinweis:** Das Master RST-FF sowie das Slave RST-FF sind taktzustandsgesteuert. Durch die Negierung in der Taktleitung reagieren sie auf unterschiedliche Taktzustände. Da, um mit den Eingängen die Ausgangszustände zu manipulieren, erst der Master und danach der Slave aktiv werden müssen, ist die Reihenfolge der benötigten Taktzustände fest und das „Vorzeichen“ der Taktflanke bestimmt. In unserem Fall wird für den Informationstransport durch das JK-MS-FF der negative Taktzustand ( $\overline{T} = 1$ ) gefolgt vom positiven ( $\overline{T} = 0$ ) benötigt. Liegt der negative Taktzustand ( $\overline{T} = 1$ ) vor, kann durch Hin- und Herschalten der Eingänge die im Master gespeicherte Information verändert werden, der Ausgangszustand des gesamten FFs wird jedoch durch die Belegung der Eingänge erst im Moment des Übergangs zum positiven Taktzustand ( $\overline{T} = 0$ ) festgelegt. Im Moment der negativen Taktflanke (positive Flanke von  $\overline{T}$ ) wird jedoch der Ausgangszustand des JK-MS-FF nicht durch die Stellung der Eingänge beeinflusst.

**Achtung:** Auf den Anschlusskappen des JK-MS-FFs sind die parallelen Eingänge fälschlicherweise mit Preset ( $P$ ) statt mit  $\overline{\text{Preset}}$  ( $\overline{P}$ ) und Clear ( $C$ ) an Stelle von  $\overline{\text{Clear}}$  ( $\overline{C}$ ) beschriftet. Gleiches gilt auch für den Takteingang.

### Fragen

- Wie entsteht der verbotene Zustand des RS-Flip-Flops? Warum ist er verboten?
- Wie kann der verbotene Zustand beim RST-Flip-Flop umgangen werden?
- Was versteht man unter dem „Prellen“ eines Schalters und wie kann es umgangen werden?
- Was sind die besonderen Eigenschaften des JK-MS-FF?
- Wie wird beim JK-MS-FF der verbotene Zustand verhindert?
- Wozu wird die Taktleitung zwischen Master und Slave invertiert und was bewirkt man damit?
- Was ist der Unterschied zwischen Taktzustandssteuerung und Taktflankensteuerung?
- Um welche Art von Taktsteuerung würde es sich handeln, falls die Negierung in der Taktleitung zwischen Master und Slave nicht vorhanden wäre?

## 5 Schieben, Multiplizieren und Rotieren

### Unterschiede: Serielle- Parallele- Verarbeitung

Größere Datenmengen werden zur Bearbeitung normalerweise in Blöcke zerlegt, welche entweder in verschiedenen Kanälen gleichzeitig (parallel) oder in einem einzigen Kanal nacheinander (seriell) verarbeitet werden. Bei seriellen Verfahren kommt man im allgemeinen mit deutlich weniger Bauteilen aus, jedoch ist die Geschwindigkeit stark begrenzt. Um zwischen den Verfahren wechseln zu können, werden Seriell-Parallel-Wandler (und

entsprechen auch Parallel-Seriell-Wandler) benötigt. Das Schieberegister der nächsten Aufgabe stellt einen einfachen Seriell-Parallel-Wandler da. Am D-Eingang können Daten nacheinander (seriell) eingegeben werden und an den LEDs gleichzeitig (parallel) ausgelesen werden.

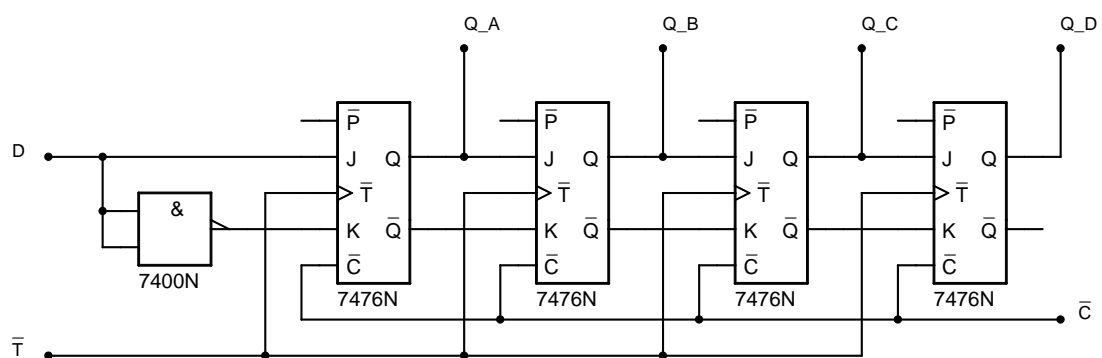
### 5.1 4-Bit-Schieberegister

Ein  $n$ -Bit-Schieberegister besteht aus  $n$  hintereinander geschalteten JK-MS-FFs bei denen jeweils **J** auf **Q** des vorangehenden Flip-Flops angeschlossen ist. Analog wird immer **K<sub>m</sub>** auf **Q<sub>m-1</sub>** geklemmt. Somit werden bei jeder positiven Taktflanke (entspricht der negativen Flanke von  $\bar{T}$ ) Informationen genau ein JK-MS-FF weitergeschoben. Der Kill-Eingang des ersten JK-MS-FFs sollte über ein *NOT* mit dessen Jump-Eingang verknüpft werden, so dass als Eingabeinstrument ein einfacher Schalter oder Taster ausreichend ist (siehe Data-Flip-Flop). Da es hier jedoch leicht zum Pellen des Schalters und damit zum versehentlichen Weiterschieben gleich um mehrere Stellen kommen kann, sollte entweder der Schalter entprellt, oder für das Taktsignal der Taktgeber des Schaltboards verwendet werden:



Über den Ausgang  $F/16$  gibt das Experimentierboard ein Taktsignal mit einer Frequenz von ca. 1 Hz aus.

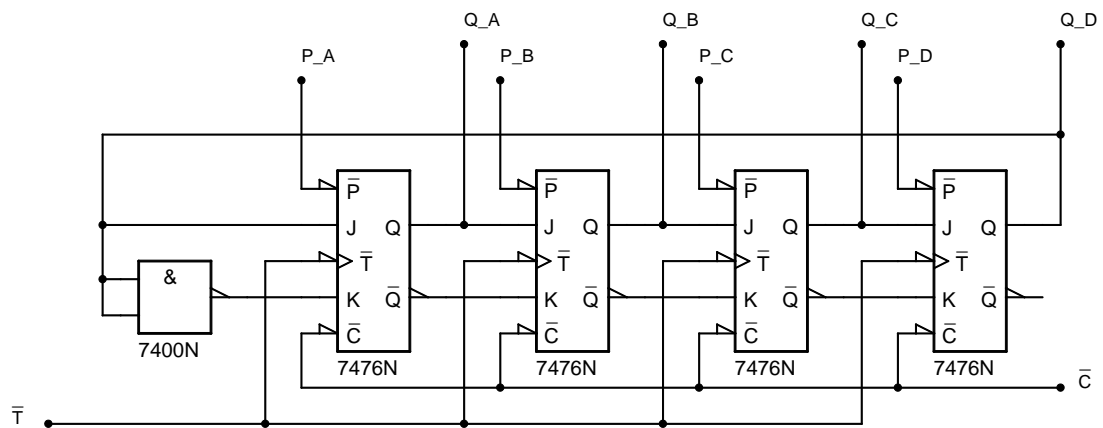
Über die Preset- ( $\bar{P}$ ) und Clear-Eingänge ( $\bar{C}$ ) können parallel die im Register gespeicherten Informationen verändert werden. Fasst man die im Schieberegister gespeicherte Information als Binärzahl auf, so kann man multiplizieren und dividieren. Bei einer Binärzahl ist der  $n$ -ten Stelle der Wert  $2^{n-1}$  zugeordnet. Daher entspricht das Schieben in Richtung des höchstwertigen Bits einer Multiplikation mit 2 und das Schieben in Richtung des niederwertigsten Bits einer Division durch 2. Mit dieser Schaltung ist jedoch nur das Schieben in eine Richtung möglich (im Schaltbild nach rechts).



### 5.2 4-Bit-Rotationsregister

Koppelt man den Ausgang des letzten JK-MS-FFs eines Schieberegisters  $Q_n$  auf den Eingang D rück, so hat man das Schieberegister in ein Rotationsregister umgewandelt.

Bei jeder positiven Taktflanke (entspricht der negativen Flanke von  $\bar{T}$ ) werden die Informationen im Kreis um eine Stelle weitergeschoben. Mit den Set- und Preset-Eingängen können nun parallel Werte für  $Q_A Q_B Q_C Q_D$  gesetzt werden.



Rotationsregister finden unter Anderem in Laufflichtern Verwendung, da hier eine gespeicherte Bitfolge endlos wiederholt ausgegeben werden muss.

**Hinweis:** Dass das Rotationsregister sich besonders gut zur Seriell-Parallel-Wandlung eignet, ist falsch, da es gar keinen seriellen Eingang hat. Das Schieberegister hingegen erfüllt mit seinem seriellen Eingang und parallelen Ausgang diese Aufgabe sehr gut.

## Fragen

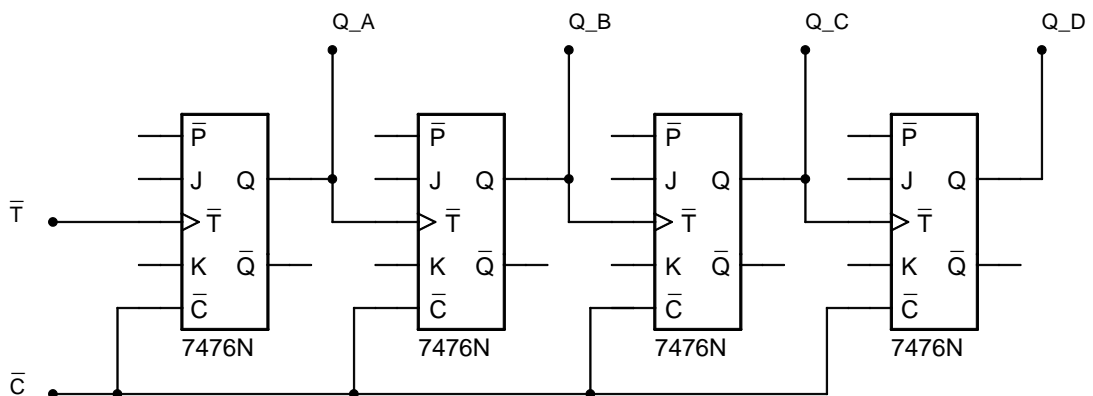
- Wozu lässt sich das Schieberegister verwenden?
- Welches Problem kann auftreten, wenn das Taktsignal über einen gewöhnlichen Schalter eingegeben wird?
- Lässt sich das Rotationsregister auch ohne die *NOT*-Verknüpfung am ersten JK-MS-FF verwirklichen?

## 6 Zähler

### 6.1 4-Bit-Asynchrone Zähler

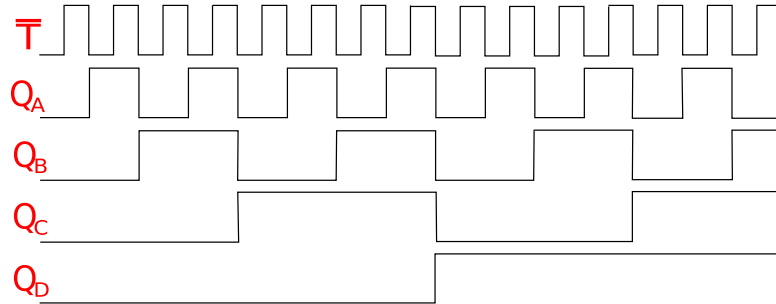
Clk	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
<b>10</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Ein Asynchrone Zähler besteht aus mehreren hintereinander geschalteten JK-MS-Flip-Flops, bei denen jeweils der Ausgang  $Q$  an den  $\bar{T}$ -Eingang des nachfolgenden angeschlossen wurde. Die Jump- und Kill-Eingänge aller Flip-Flops sind offen, folglich liegt an ihnen  $1$ , somit arbeiten alle FFs im Toggle-Betrieb. Da jedes JK-MS-FF taktflankengesteuert ist, werden an jedem Flip-Flop zwei Änderungen des Taktes benötigt um eine Änderung des Ausgangs herbeizuführen. Es werden also am  $n$ -ten JK-MS-FF zwei negative Flanken von  $\bar{T}_n$  benötigt um in den negierten Takteingang des  $(n+1)$ -ten  $\bar{T}_{n+1}$  FFs eine negative Flanke einzugeben. Hierdurch halbiert sich nach jedem Flip-Flop der Takt. Diese Verfahren ist die einfachste Art von Zähler, es wird jedoch, da jede Stufe mit einem anderen Takt arbeitet, **asynchron** gezählt.



Dies bringt Probleme mit sich: Da jedes Flip-Flop eine gewisse Zeit zum Umschalten braucht, wird der Zähler mit jeder weiteren Stufe immer ungenauer, da sich der Takt

nicht nur halbiert, sondern sich noch zusätzlich in jeder Stufe die Laufzeit durch ein JK-MS-FF hinzuaddiert. Um einen präzisen Zähler mit sehr vielen Stufen zu bauen ist dieses Verfahren daher gänzlich ungeeignet.



An obigem Schaubild lässt sich sehr gut die Flankensteuerung der JK-MS-FFs erkennen. Diese reagieren nur auf positive Taktflanken, was negativen Flanken von  $\overline{T}$  entspricht (siehe Abschnitt: JK-MS-FF). Beim Hintereinanderschalten mehrerer JK-MS-FFs ist es viel sinnvoller immer nur das  $\overline{T}$ -Verhalten zu betrachten, da wir den  $Q$ -Ausgang jeder Stufe ja an den  $\overline{T}$  Eingang der nächsten anschließen. Leicht lässt sich erkennen, dass jede Stufe mit der halben Frequenz der vorangehenden Stufe arbeitet.

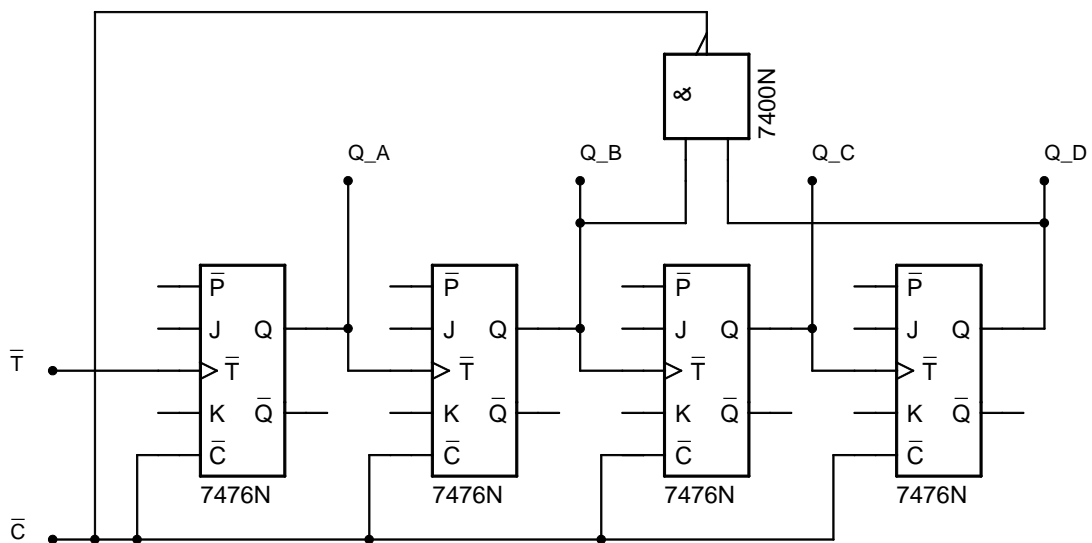
**Hinweis:** Der Asynchrone Zähler reagiert nur auf negative Flanken von  $\overline{T}$ . Positive Flanken von  $\overline{T}$  führen zu **keiner** Änderung des Ausgangszustandes.

## 6.2 Asynchroner Dezimalzähler

Ein binärer 3-Bit-Zähler zählt von 0 bis 7, also 8 Stufen, ein binärer 4-Bit-Zähler zählt von 0 bis 15, also 16 Stufen. Um somit mit einem binären Zähler dezimal zählen zu können, wird ein 4-Bit-Zähler benötigt der beim Erreichen der Zahl 10 auf 0 zurückgesetzt wird. Zuerst muss man anhand obiger Tabelle ablesen wie die dezimale Zahl 10 binär codiert wird:

$$\text{Dezimal: } 10 \quad \Longleftrightarrow \quad \text{Binär: } 1010$$

Da wir nur vorwärts zählen ist es völlig ausreichend, das viert-niedrigste und das zweit-niedrigste Bit mit *NAND* zu verknüpfen und dies auf die Clear-Eingänge der JK-MS-FFs zu geben (Achtung: Die Clear-Eingänge sind invertiert  $\overline{C}$ , daher *NAND* statt *AND*). Es lässt sich einfach die vorige Schaltung um diese Verknüpfung erweitern.

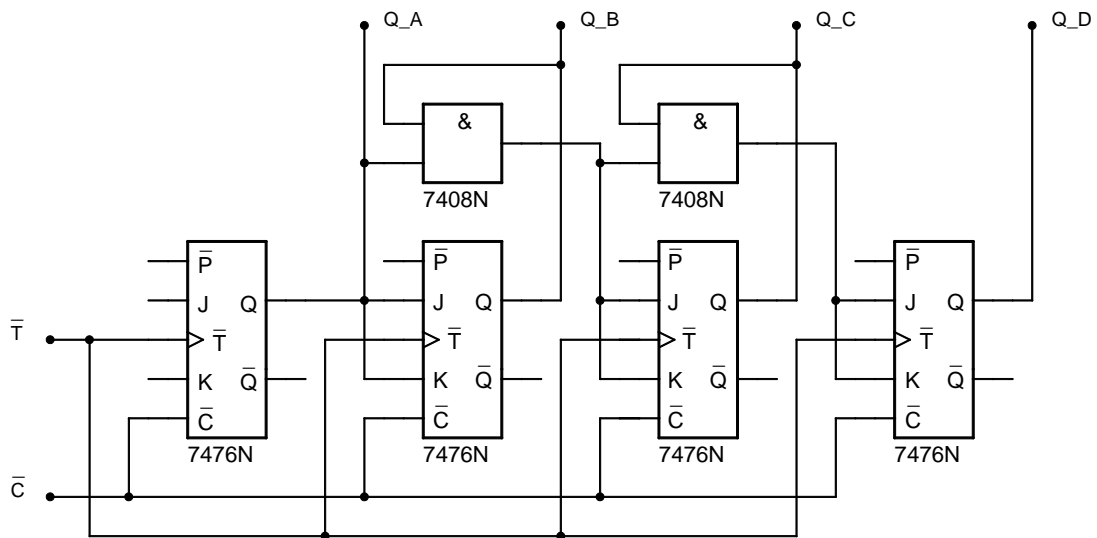


Bei einem Zähler, welcher sowohl vorwärts wie rückwärts zählen soll, wäre dies nicht ausreichend. Hier müssten wir das niedrigste und das dritt-niedrigste Bit *NOR* verknüpfen, sowie das zweit-niedrigste, das viert-niedrigste und den Ausgang dieser *NOR*-Verknüpfung mit *NAND* verknüpfen und auf die Clear-Eingänge geben. So würde auch beim Abwärtszählen genau bei der Dezimalzahl 10 resettet werden.

Falls die Synchronzähler nicht aufgebaut werden, empfiehlt es sich den Aufbau stecken zu lassen und direkt mit Aufgabe 7 weiterzumachen, da sie sehr schnell durchzuführen ist.

### 6.3 4-Bit-Synchronzähler

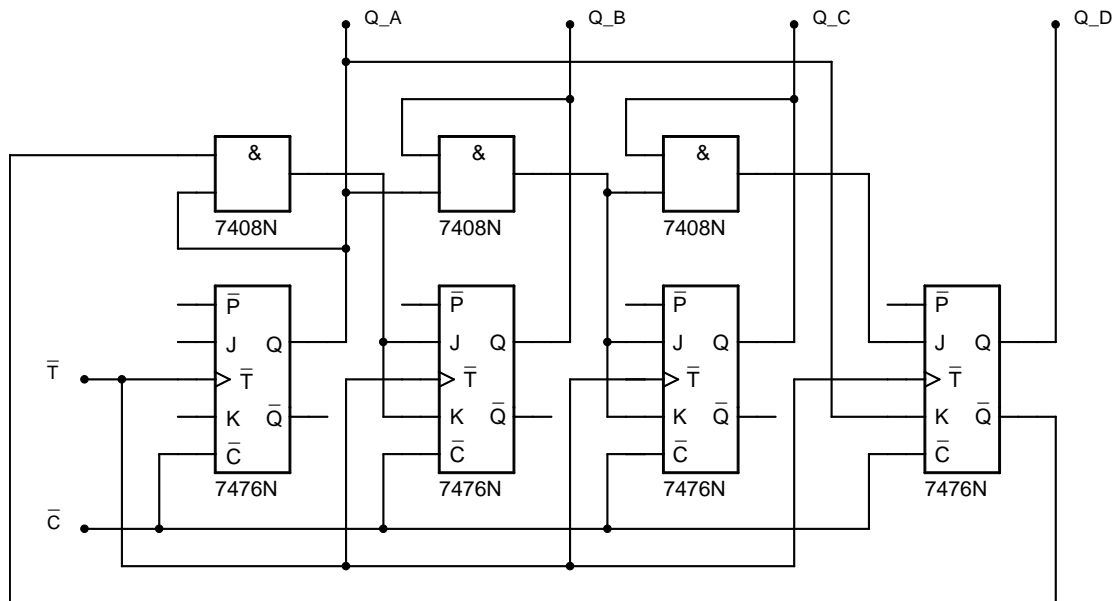
Um einen präzisen Zähler zu verwirklichen, müssen alle Stufen mit dem gleichen Takt betrieben werden, daher **Synchronzähler**. Um das binäre Zählen zu gewährleisten, muss eine Stufe immer mit der halben „Geschwindigkeit“ laufen wie die vorangehende (vom niedrigsten zum höchsten Bit aus gesehen). Dies lässt sich erreichen, indem man an jeder Stufe die Ausgänge aller vorangehenden Stufen mit *UND* verknüpft und dies an **J**ump und **K**ill anschließt um das Toggeln zu aktivieren. Hierdurch erhält diese Stufe das gleich Taktsignal wie alle anderen (synchron), toggelt aber nur dann, wenn alle vorangehenden gerade auf **1** stehen.



Der Schaltungsaufwand lässt sich reduzieren, indem man einfach die Ausgänge der letzten und vorletzten Stufe *UND* verknüpft und diese an **Jump** und **Kill** anlegt. Jump und Kill der zweit-niedrigsten Stufe kann direkt, ohne logische Verknüpfung an den Ausgang der niedrigsten angeschlossen werden. Hierdurch erreicht man das gleiche, wie im vorigen Vorschlag, nur sind so die Verknüpfungen gestaffelt. Bei sehr schnellen und sehr langen Zählern kann es jedoch durch die von den Schaltzeiten der *UND*-Verknüpfung resultierenden Verzögerungen zu Fehlern kommen. Normalerweise (und auch hier im Praktikum) ist dies jedoch kein Problem.

## 6.4 Synchroner Dezimalzähler

Der Synchronzähler ließe sich, analog zum Asynchronzähler in 6.2, sehr leicht zum Dezimalzähler erweitern, indem man das höchste und das zweitniedrigste Bit *UND*-verknüpft und dies auf die Clear-Eingänge der Flip-Flops gibt. Diese Vorgehensweise ist für die erfolgreiche Durchführung des Versuchs vollkommen ausreichend. Eine viel elegantere Lösung ist jedoch den invertierten Ausgang des höchsten Bits  $\overline{Q}_D$  mit dem Ausgang des niedrigsten  $Q_A$  *UND* zu verknüpfen und dies auf Jump  $J_B$  und Kill  $K_B$  des zweitniedrigsten Flip-Flops zu geben. Außerdem wird  $Q_A$  auf den Kill-Eingang des höchsten Bits  $K_D$  gelegt. Beim Zählen von 0 bis 7 hat dies keine Auswirkung, da das höchsten Bit hier **0** ist und somit sein invertierter Ausgang die ganze Zeit auf **1** liegt. Da **A AND 1**  $\Leftrightarrow$  **A** wird in diesem Bereich genau so wie beim normalen Synchronzähler gezählt. Der Übergang auf die dezimale 8 ist ebenfalls kein Problem, es liegt vom dritt-niedrigsten Bit **1** an Jump  $J_D$  und von der gerade beschriebenen Verknüpfung ebenfalls **1** an Kill  $K_D$  des höchsten Bits an, dieses Bit wird also getoggelt. Nun ist das höchsten Bit **1** und an seinem invertierenden Ausgang  $\overline{Q}_D$  liegt **1**. Die gerade beschriebene Verknüpfung ist nun: **A AND 0**  $\Leftrightarrow$  **0**. Somit kann nur noch das niedrigste Bit beim nächsten Takt toggeln (ergibt die 9), das zweitniedrigste jedoch nicht mehr. Stattdessen wird beim Erreichen der 9 auch über den Ausgang dieser Verknüpfung eine **1** auf Kill des höchsten Flip-Flops  $K_D$  übertragen, da an dessen Jump  $J_D$  aber noch **0** anliegt, wird dieses Bit auf **0** gesetzt. Somit springt der Zähler von der 9  $\leftrightarrow$  1001 im nächsten Takt auf 0  $\leftrightarrow$  0000 zurück.



Diese Anschlussvariante würde also auch ohne Clear-Eingänge auskommen. Es empfiehlt sich die Schaltung aufgebaut zu lassen, da sie sehr leicht in den Aufbau aus Aufgabe 7 erweitert werden kann.

### Fragen

- Was ist beim Asynchrone Zähler asynchron?
- Wie könnte man den Asynchrone Zähler rückwärts zählen lassen (angenommen vor Eingabe des Taktsignals seien alle FFs über  $P_A \dots P_D$  auf 1 gesetzt worden)?
- Wozu wird das NAND beim asynchronen Dezimalzähler benötigt?
- Wie könnte man aus dem asynchronen Zähler (6.1) einen Oktalzähler machen?
- Wozu benötigt man die ANDs beim synchronen 4-Bit-Zähler?
- Wie viele ANDs bräuchte man für einen synchronen 5-Bit-Zähler und wo würde man die/das zusätzliche einbauen?

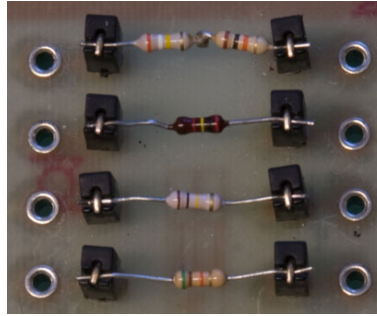
## 7 Digital-Analog-Wandlung

Die am Dezimalzähler aus (6.2 bzw. 6.4) digital ausgegebene Zahl ( $0 \dots 9$ ) soll nun mittels eines Drehspulenmessinstruments analog gewandelt werden. Der Ausschlag der Nadel des Messgerätes soll also proportional zur aktuell am Zähler angezeigten Zahl sein. Gemessen wird im  $\mu A$ -Bereich wobei ein Maximalstrom von  $90 \mu A$  nicht überschritten werden darf (90% von  $100 \mu A$ ). Dieser Strom soll bei Erreichen der höchsten Stelle (9) fließen. Daraus folgt, dass sich in jedem Zählschritt der Strom um  $90 \mu A / 9 = 10 \mu A$  erhöhen soll. Ist also die Einerstelle des Zählers auf 1, sollen  $10 \mu A$  durch einen noch zu berechnenden, daran

angeschlossenen Widerstand  $R_1$  fließen. Bei einer Ausgangsspannung der ICs von ca. 4 V entspricht das einem Widerstand von:

$$R_1 = \frac{U}{I} = \frac{4 \text{ V}}{10 \mu\text{A}} = 400 \text{ k}\Omega$$

An der Zweierstelle  $Q_B$  soll der doppelte Strom fließen, demnach muss der hier angeschlossene Widerstand halb so groß sein wie der an  $Q_A$ :  $R_2 = 4 \text{ V}/20 \mu\text{A} = 200 \text{ k}\Omega$  usw. für die höheren Stellen. Auf dem Experimentierboard sind die benötigten Widerstände bereits in einer Reihe aufgebaut:

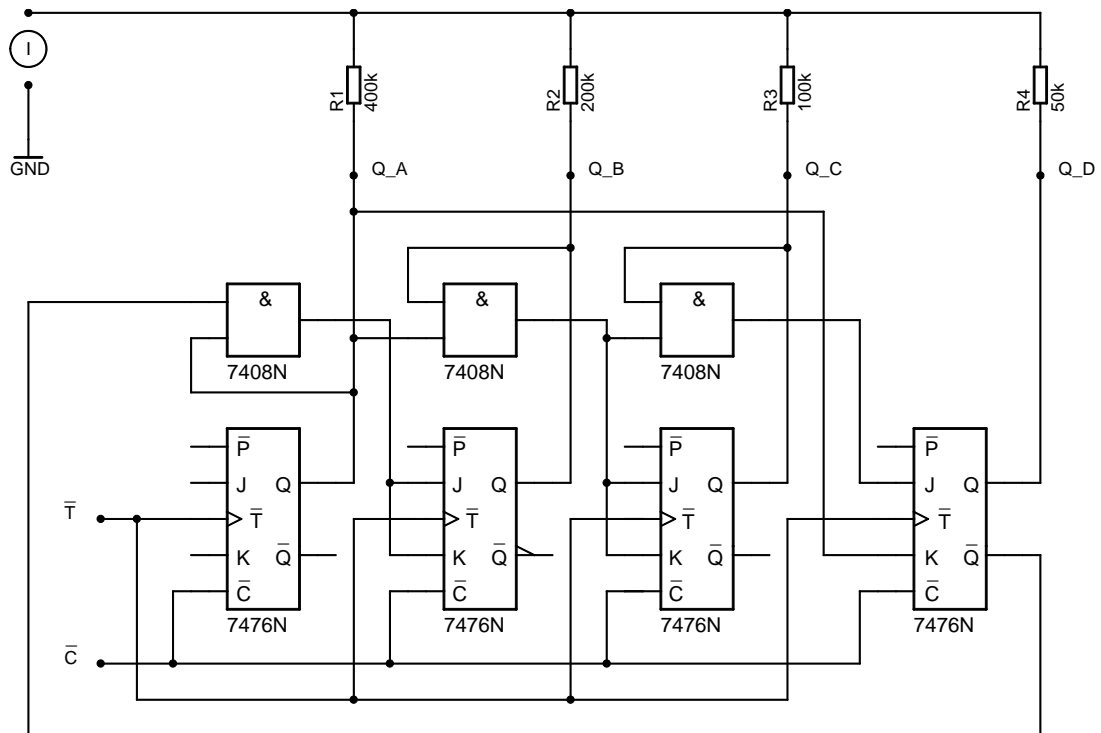


Mit dem Multimeter lässt sich leicht der Widerstand dieser bestimmen:

Widerstand	Strom	Bit	Binärzahl	Dezimalzahl
400 k $\Omega$	10 $\mu\text{A}$	niedrigstes	0001	1
200 k $\Omega$	20 $\mu\text{A}$	zweitniedrigstes	0010	2
100 k $\Omega$	40 $\mu\text{A}$	drittniedrigstes	0100	4
50 k $\Omega$	80 $\mu\text{A}$	viertniedrigstes	1000	8

Somit lässt sich Digital-Analog-Wandlung sehr einfach durchführen, indem man den Ausgang des niederwertigsten Bits an den größten (bei uns den obersten), jedes weitere Bit an den nächst kleineren Widerstand anschließt. Die anderen Enden der Widerstände schließt man parallel an den Pluspol des Multimeters an, den Minuspol an Masse. Nun repräsentiert der Strom durch das Multimeter die analoge „Übersetzung“ der digitalen Zahl des Zählers.

Generell stellt man fest, dass der Maximalstrom nur knapp 75  $\mu\text{A}$  anstelle der erwarteten 90  $\mu\text{A}$  entspricht. Dies liegt daran, dass die Spannungsversorgung der Experimentierboards niedriger ist als im Aufgabentext angegeben. Daher ist auch die Ausgangsspannung der ICs niedriger als 4 V. Die Stromzunahme ist mit ca. 7,5  $\mu\text{A}$  pro Zählschritt dennoch fast linear.



Durch die großen Unterschiede der Widerstände begrenzen deren Fehler die Genauigkeit des DA-Wandlers. Da die Widerstände immer prozentuale Abweichungen aufweisen, sind die Fehler des größten Widerstandes in der Größenordnung des kleinsten Widerstandes. Daher verwendet man normalerweise keine DA-Wandler mit stark unterschiedlichen Widerständen, stattdessen kann man durch Aufteilen des Stromes auch mit gleichen Widerständen sehr große Bereiche abdecken ( $\Rightarrow$  R2R-Kette).

### Fragen

- Warum können wir nicht einfach alle Ausgänge  $Q_A \dots Q_D$  verbinden und dann die Ausgangsspannung messen?
- Addieren sich die Ströme der einzelnen Stufen oder ihre Spannungen?
- An welche LED muss der größte Widerstand angeschlossen werden?
- Wieso müssen die Widerstände von einer zur nächsten Stufe immer halbiert werden?
- Liest überhaupt irgend jemand diese Fragen?

### Schlusswort

Vielen Dank an Bastiaan Hovestreydt für die ausführliche Fallunterscheidung beim Subtrahierer und auch an Herrn Dr. Blüm für das Korrekturlesen.

Kritik, Verbesserungsvorschläge und Korrekturen bitte an [mail@christian-benz.de](mailto:mail@christian-benz.de) oder [mail@christianbarth.com](mailto:mail@christianbarth.com) schicken.